MICROSOFT®

Microsoft® Excel

Complete Spreadsheet with Business Graphics and Database

Arrays, Functions, and Macros

Microsoft® Excel Arrays, Functions, and Macros

Microsoft® Excel

Arrays, Functions, and Macros

for the Apple® Macintosh™

Microsoft Corporation

Information in this document is subject to change without notice and does not represent a commitment on the part of Microsoft Corporation. The software described in this document is furnished under a license agreement or nondisclosure agreement. The software may be used or copied only in accordance with the terms of the agreement. It is against the law to copy Microsoft Excel on magnetic tape, disk, or any other medium for any purpose other than the purchaser's personal use.

© Copyright Microsoft Corporation, 1985

If you have comments about this documentation or the software, complete the Software Problem Report at the back of this manual and return it to Microsoft.

Microsoft, the Microsoft logo, and Multiplan are registered trademarks, and The High Performance Software is a trademark, of Microsoft Corporation.

Apple is a registered trademark, and the Apple logo and Imagewriter are trademarks, of Apple Computer, Inc.

Macintosh is a trademark licensed to Apple Computer, Inc.

The Finder, Imagewriter Driver, System, and Switcher are copyrighted programs of Apple Computers, Inc. licensed to Microsoft Corporation to distribute for use only in combination with Microsoft's software for the Apple Macintosh computer.

APPLE COMPUTER, INC. MAKES NO WARRANTIES, EITHER EXPRESS OR IMPLIED, REGARDING THE ENCLOSED COMPUTER SOFTWARE PACKAGE, ITS MERCHANTABILITY, OR ITS FITNESS FOR ANY PARTICULAR PURPOSE. THE EXCLUSION OF IMPLIED WARRANTIES IS NOT PERMITTED BY SOME STATES. THE ABOVE EXCLUSION MAY NOT APPLY TO YOU. THIS WARRANTY PROVIDES YOU WITH SPECIFIC LEGAL RIGHTS. THERE MAY BE OTHER RIGHTS THAT YOU MAY HAVE WHICH VARY FROM STATE TO STATE.

Document Number: 690560030-100-R01-1285

Contents

Arrays 1

1 Using Arrays 5
Array Formulas 8
Array Constants 11
Editing and Evaluating Array Formulas 12
Array Arguments and Results 13
Array Functions 16
Arrays and External References 17
Arrays and Charting 18

Functions 21

2 Introduction to Functions 25
What Is a Function? 25
Syntax of Functions 26
Entering Functions into Formulas 27

3 Worksheet Function Directory 29
Arguments to Functions 29
Functions by Subject Category 33
Mathematical Functions 33
Statistical Functions 34
Database Functions 35
Trigonometric Functions 35
Logical Functions 36
Text Functions 36
Financial Functions 37
Date Functions 37
Special-Purpose Functions 38
Alphabetical Listing of Worksheet Functions 39

Macros 77

4 Introduction to Macros 81
Macros 81
The Excel Macro Language 82
Macro Sheets 83

5 Command Macros 85
What Is a Command Macro? 85
Entering a Command Macro 86
Naming a Command Macro 88
Running a Command Macro 89
A Sample Command Macro 90
Editing a Macro 93
Smart Macros 94
Making Command Macros Interactive 95
Calculating Macros 98
Branching with the GOTO Function 98
Branching with the ref Function 99
Writing a Command-Equivalent Macro 100

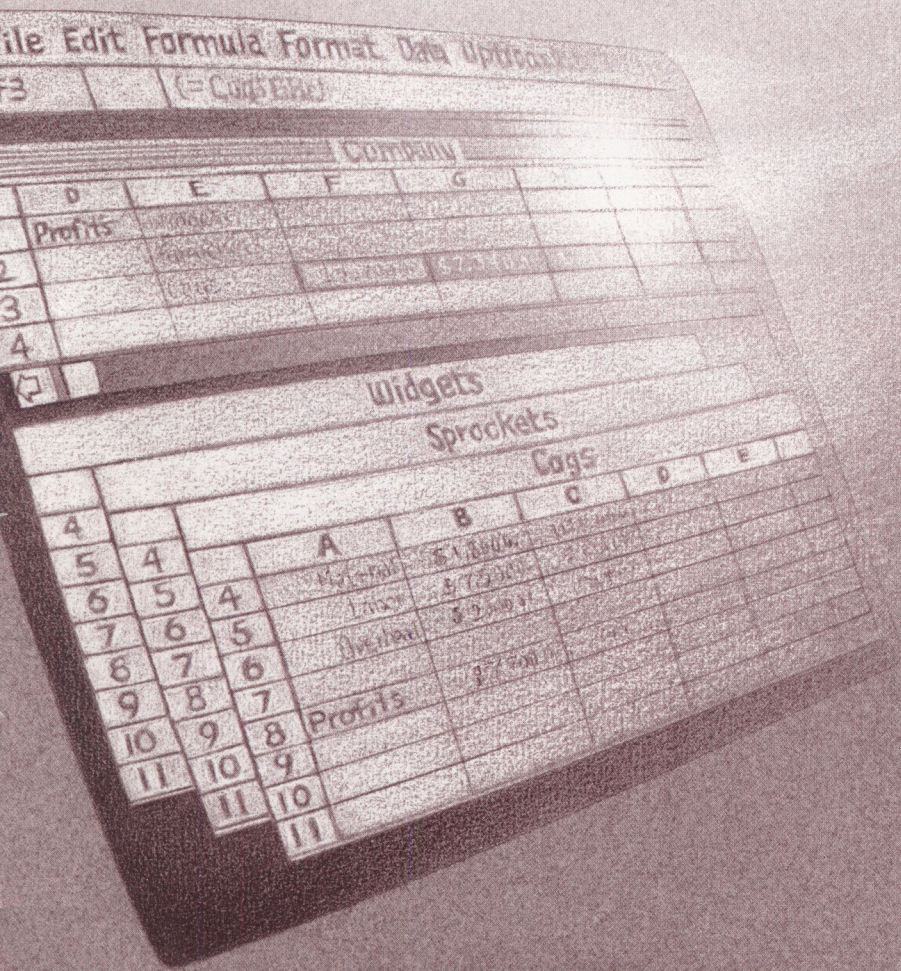
6 Function Macros 103
What Is a Function Macro? 103
Entering a Function Macro 104
Naming a Function Macro 105
Using a Function Macro 105
A Sample Function Macro 106
Enhancing Function Macros 109
The ARGUMENT and RESULT Functions 111
Using Arrays in Function Macros 113

7 More Information about Macro Sheets 115
Setting the Recorder Range for a Command Macro 115
Starting, Stopping, and Restarting the Recorder 116
Using Relative Record and Absolute Record 117
Referencing the Active Worksheet 117
Organizing Your Macro Sheets 118
Using Labels and Comments in Macros 119
Calling and Running Macros 121
Macro Sheet Calculation 123
How Excel Calculates Macros 124
Formulas with Action-Taking Functions 127
Controlling Calculation with Loops and Conditionals 129
Interrupting a Macro 131
Passing Arguments and Returning Results 132
Defining Names for Arguments 132
Entering Arguments in Cells 133
Argument Types 134
Result Types 135
More on Types 136

8 Macro Language Function Directory 137
Arguments to Macro Language Functions 137
Arguments to Command-Equivalent Functions 137
Macro Language Functions by Category 138
Command-Equivalent Functions by Menu 138
Action-Equivalent Functions 141
Functions that Control Macro Calculation, Arguments, and User Interaction 142
Special-Purpose Functions 143
Alphabetical Listing of Macro Language Functions 144

Index 193

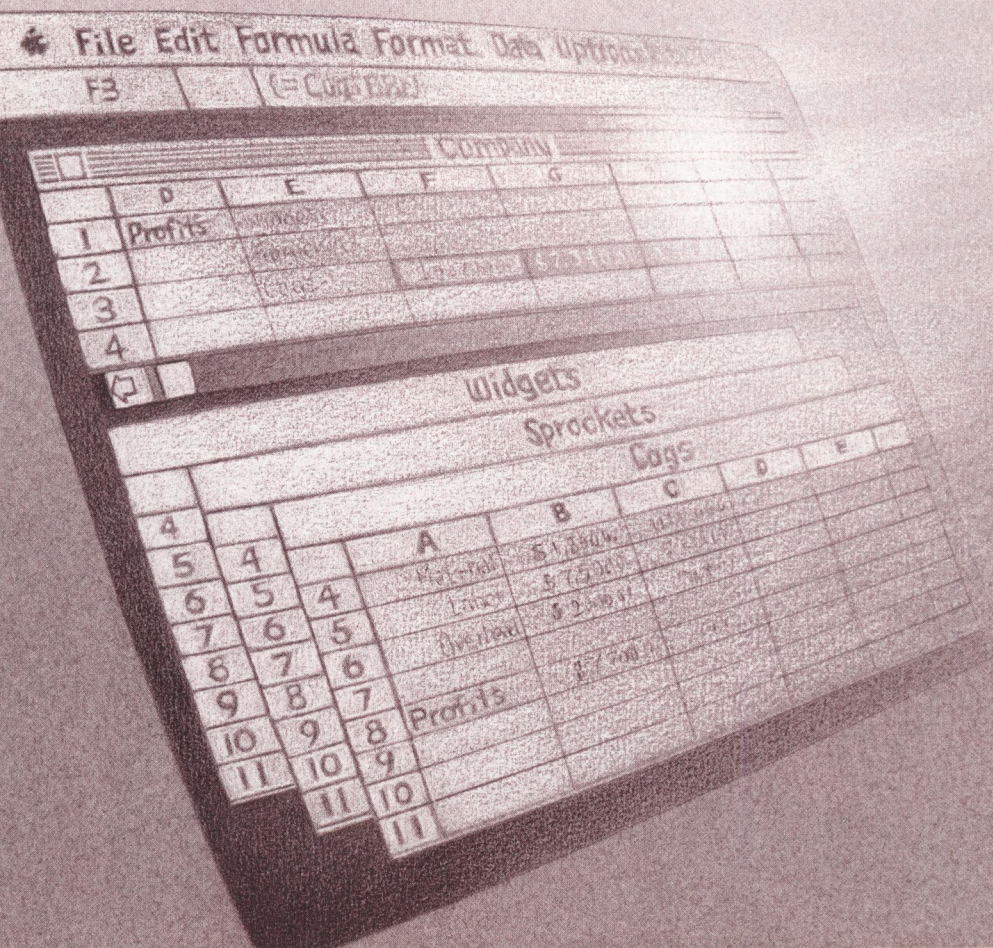
Arrays

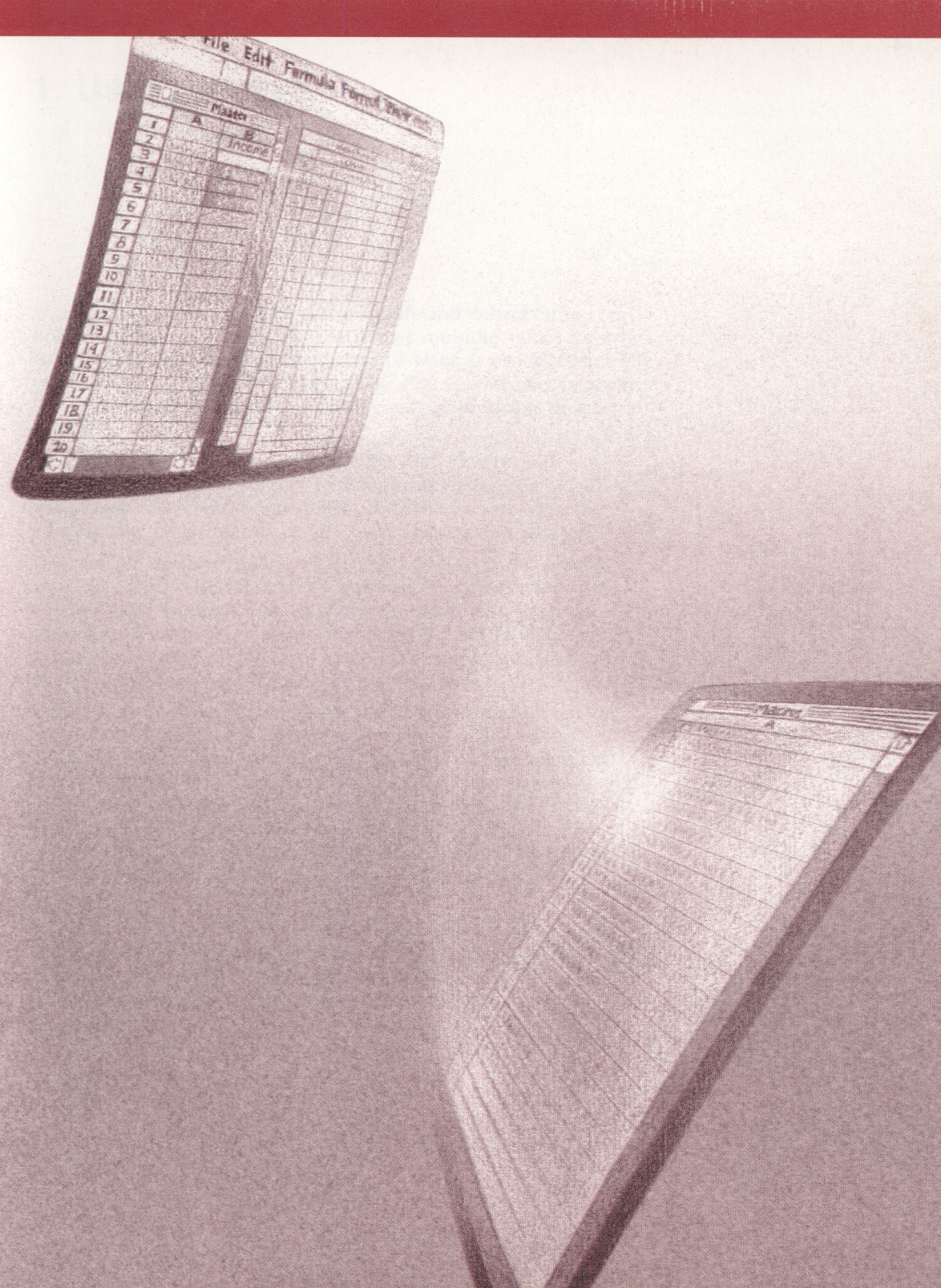


Arrays

This part of the manual introduces the concept of arrays and describes how to use arrays in Microsoft® Excel.

You will find information on using arrays in formulas, constants, and functions, as well as a discussion of arrays as arguments and results. In addition, this part explains the use of arrays in external references and charting.




File Edit Formula Format Data
Master
A
B
Income
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
Macro
A

1 Using Arrays

With Excel, you can use arrays to simplify and enhance the creation of worksheets when you want to use multiple values for arguments to functions and operators, and/or want to produce multiple values for results. Microsoft Excel is the first spreadsheet program that accepts and uses arrays as arguments and produces arrays as results.

To understand arrays, first consider the following property of every formula on a worksheet without arrays: each of the individual operations within a formula produces a single value as a result, and, consequently, so does the formula as a whole.

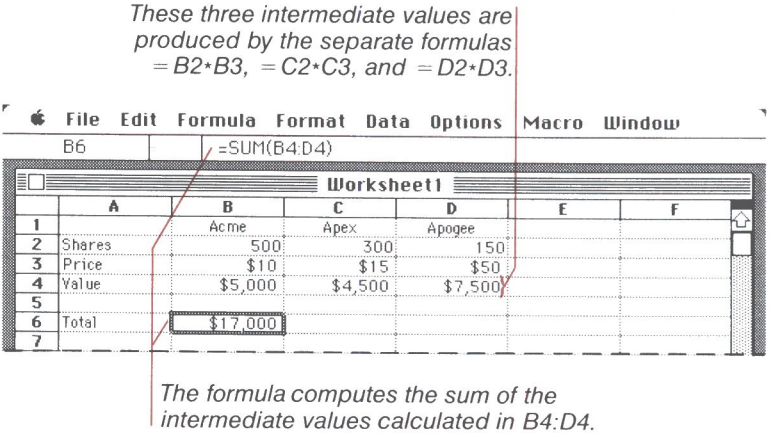
For example, in the formula =PV(15%,5,A1)+A3, the percent operator acts on the value 15 to produce the single value 0.15, which the PV function then operates on, along with the value 5 and the reference A1, to produce yet another single value. Finally, the addition operator adds that value to the reference A3 to produce a single value as the result of the formula.

Notice that all of the arguments in the above formula are single values or references to single cells containing single values. In worksheets without arrays, most functions and all operators accept only single values as arguments. A few functions have arguments which can be references to a range of cells, rather than to just a single cell. For example, in the formula =SUM(A1:A3), the argument A1:A3 is a reference to three cells containing three values. Even these functions, however, produce single values as results.

There are few times when the limitation to single values would actually prevent you from creating a worksheet model. However, there are times when it would be much simpler and faster to build a formula whose operations produced multiple values as results.

Creating a model the hard way — without arrays

For example, suppose you want to create a worksheet model to compute the total value of your holdings in a number of different stocks. You might have 500 shares of one stock trading at $10 a share, 300 shares of another trading at $15, and 150 shares of a third stock at $50. The total value of your holdings is the sum of the values of each stock. The value of each stock is the product of the price of each and the number of shares of each. So, to compute the total value of your holdings, you might create the following worksheet model:

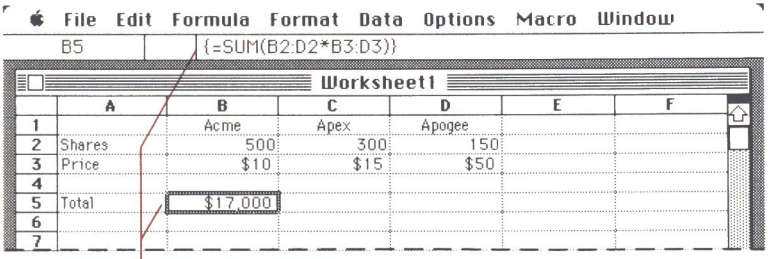

These three intermediate values are produced by the separate formulas =B2*B3, =C2*C3, and =D2*D3.
File Edit Formula Format Data Options Macro Window
B6
=SUM(B4:D4)
Worksheet1
A B C D E F
1 Acme Apex Apogee
2 Shares 500 300 150
3 Price $10 $15 $50
4 Value $5,000 $4,500 $7,500
5
6 Total $17,000
7
The formula computes the sum of the intermediate values calculated in B4:D4.

You enter the numbers of shares 500, 300, and 150 in cells B2:D2, and the prices $10, $15, and $50 in cells B3:D3. You then compute the value of each stock by entering the formula =B2*B3 in cell B4, the formula =C2*C3 in C4, and the formula =D2*D3 in D4. These formulas produce three single values: $5,000, $4,500, and $7,500. Then, in another cell, you enter the formula =SUM(B4:D4), whose result is the single value $17,000.

In the above model, you had to enter three separate formulas to compute the values of the three stocks, and an additional formula to sum those three values — four formulas in all.

Creating a model the easy way — with arrays

Instead, with Excel you can compute the total value of your holdings with one formula: =SUM(B2:D2*B3:D3). Here, you want to multiply the number of shares in each cell of B2:D2 by the corresponding price in each cell of B3:D3, and then sum the three resulting products.



The array formula computes the array of intermediate values, and then sums the values in the array.

What is an array?

B2:D2 contains three values rather than a single value. These three values are referred to as an array. An array is a collection of single values organized into a rectangle with a specified number of rows and columns. The array of values contained in B2:D2 is organized into one row and three columns and is referred to as a 1 by 3 array. Similarly, the three values $5,000, $4,500, and $7,500 that result from multiplying B2:D2 and B3:D3 constitute an array.

In spreadsheet programs that do not support arrays, the above formula would produce an error, because the multiplication operator operates on single values and produces a single value as a result. Excel, however, understands and operates on arrays as arguments and produces arrays as results.

Array Formulas

Entering an array formula

To enter an array formula in Excel, type or paste the formula into the formula bar, and hold down the Command key when you press the Enter key or click the enter box. When you do this, you are telling Excel that the formula contains an array as an argument and/or produces an array as the result of an operation. A formula entered with the Command key held down is called an array formula.

If the active cell contains an array formula, then, when the formula bar is inactive, the array formula is enclosed in braces ({}), just as is the case with a table. This makes it easy to tell if a cell contains an array formula. Do not type the braces; Excel provides them automatically.

In an array formula, you can give an array as an argument to a function or operator that normally operates on single values and produces a single value as a result, such as the multiplication operator. Excel performs the function or operation on each single value within the array to produce an array as a result. When you give multiple arrays as arguments, Excel performs the function or operation on corresponding successive values of the arrays.

For example, you can enter the formula =SUM(B2:D2*B3:D3) in a single cell by holding down the Command key when you press the Enter key or click the enter box. Excel multiplies each number of shares in B2:D2 by the corresponding price in B3:D3 to produce the array of products $5,000, $4,500, and $7,500, and then sums the values in the array.

In the above example, while the operation of multiplying B2:D2 and B3:D3 produces an array as a result, the SUM function sums the values in the array to produce a single value as the result of the entire array formula.

Using array formulas to get arrays as results

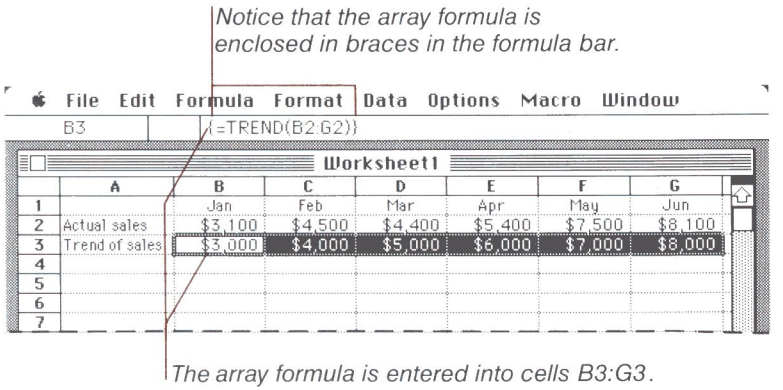
Most results you want to compute on a worksheet are described by single values. However, some results can only be described by multiple values.

For example, suppose you want to compute the trend of your monthly sales figures over the last six months. That trend is described not by one value but by six — the sales in each of the last six months on the line that best fits the actual sales data. While it is possible to create a worksheet model that uses a separate set of formulas to approximate your sales for each month, it is preferable to build one formula that produces an array of six values. Because each cell on a worksheet can contain only one value, you will want to enter this formula in more than one cell at once. With Excel, you can do just that.

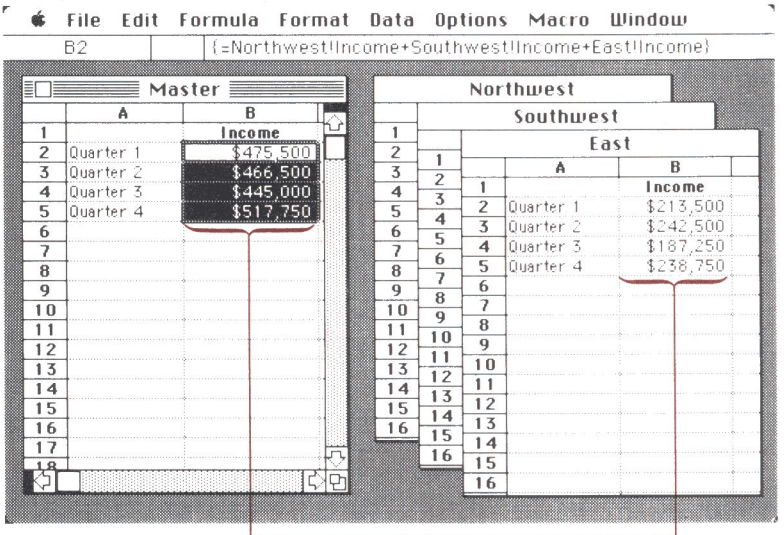
When you enter an array formula, Excel enters the formula not just in the active cell but in all of the selected cells. It places each value of the array produced by the formula in one cell of the selection. Therefore, to enter an array formula into a range of cells, you need to first select the range, then type or paste the formula, and then hold down the Command key when you enter the formula.

If you double-click a cell that contains an array formula, the entire range of cells containing the formula is selected, just as with a table. This makes it easy to see which range of cells the array formula is entered in.

For example, suppose your monthly sales figures from the last six months are entered in cells B2:G2. You can enter the formula =TREND(B2:G2) into the range of cells B3:G3 by selecting the range and holding down the Command key when you press the Enter key or click the enter box. Excel takes the 1 by 6 array of actual sales in B2:G2, produces a 1 by 6 array of approximated sales, and places each sales value in one cell of B3:G3.


Notice that the array formula is enclosed in braces in the formula bar.
File Edit Formula Format Data Options Macro Window
B3
{=TREND(B2:G2)}
Worksheet1
A B C D E F G
1 Jan Feb Mar Apr May Jun
2 Actual sales $3,100 $4,500 $4,400 $5,400 $7,500 $8,100
3 Trend of sales $3,000 $4,000 $5,000 $6,000 $7,000 $8,000
4
5
6
7
The array formula is entered into cells B3:G3.

Here is another example of an array formula entered in more than one cell. Suppose you have three worksheets for the three different regions in which your company does business. Each worksheet has a column of four cells containing the projected income for the next four quarters, and the range of cells is named Income on each worksheet. On your master worksheet, you can compute the total projected income of your company by entering, also in a column of four cells, the following array formula: =Northwest!Income + Southwest!Income + East!Income.



The arguments Northwest!Income, Southwest!Income, and East!Income are each 4 by 1 arrays. The result of the formula, shown on the Master worksheet, is also a 4 by 1 array.

Each argument in this formula is a 4 by 1 array. Excel adds the corresponding income values in each array to produce a 4 by 1 array of the total projected income for each quarter.

Array formulas are particularly useful when you want to refer to cells on another worksheet. For more information on external references, see "Linking Documents" in Chapter 8 of the *User's Guide*. See also "Arrays and External References" in this chapter.

Using Excel's Macro Language, it is possible to create your own functions that operate on arrays and/or produce arrays as results. For more information on using array formulas in functions you create with macros, see Chapter 6, "Function Macros."

Array Constants

In any formula where you use a reference to a single cell containing a single value, you can instead use the value itself as a constant.

Suppose you want to compute the future value of $100 deposited every month for 12 months in an account earning 0.75% monthly interest. You can enter the numbers 0.0075, 12, and 100 in cells A1, A2, and A3 and then use the formula =FV(A1,A2,A3). Or, you can use the formula =FV(0.0075,12,100).

Sometimes, it is preferable to enter a constant in a cell and refer to that cell in your formulas, particularly if the same constant is used in several formulas on the worksheet. At other times, you may want to enter the constant directly into a formula, without entering it in a cell.

Just as with single values, in any formula where you use a reference to an array, you can instead use the array itself as a constant.

Representing an array constant

An array constant is represented by enclosing the values in the array within braces ({}) and separating them with commas and semicolons. The values within one row of the array are separated by commas, while the rows of the array are separated by semicolons. For example, if cells A1:C1 contain the values 1, 2, and 3, and cells A2:C2 contain the values 4, 5, and 6, the 2 by 3 array constant of values contained in A1:C2 is represented as {1,2,3;4,5,6}.

In a previous example, we entered the numbers 500, 300, and 150 in cells B2:D2 and the numbers 10, 15, and 50 in cells B3:D3, and then used the formula =SUM(B2:D2*B3:D3). Instead, we could have entered the array constants directly into the formula, as in =SUM({500,300,150}*{10,15,50}).

Contents of an array constant

The values in an array constant can be numbers, text, logical values, or error values. They must be in a format that is allowed within formulas. That is, numbers can be in integer, decimal, or scientific format, but cannot contain dollar signs, parentheses, or percent signs. Text must be enclosed by double quotation marks. An array can contain values of different types. Below are some examples of constant arrays.

Type	To get this	
{1,2,3}	1 by 3 array	1 2 3
{1;2;3}	3 by 1 array	1 2 3
{1,2,3;"a","b","c"}	2 by 3 array	1 2 3 a b c

The values in an array constant must themselves be constant values and not formulas, so {1 + 2,"a"&"b"} is an invalid array constant. Also, you cannot enter arrays that have columns or rows of unequal length, so {1,2,3;4,5} is an invalid array constant.

Editing and Evaluating Array Formulas

In addition to the differing characteristics of array formulas discussed earlier, Excel treats these formulas differently from normal formulas in other ways, as well.

Editing an array formula

If you enter an array formula into a range of cells rather than into just one cell, you must treat those cells as a unit for some worksheet operations. In such cases, after you have entered the array formula, you cannot edit an individual cell within the range or choose any command that would disrupt the range of cells in any way. If you try to enter a normal formula into an individual cell, Excel displays an alert that says "Illegal operation on an array formula."

To edit an array formula, you must select either a single cell within the range, or the entire range, and then hold down the Command key when you enter the edited formula. If you select just a single cell, Excel automatically selects the entire range when you enter the edited formula.

Similarly, you cannot clear the formula of an individual cell within the range, but must select the entire range to clear the array formula. Also, you cannot delete or cut a single row or column of the range. You can, however, format or clear the format from cells in the range without selecting the entire range.

Evaluating an array formula

An array formula is evaluated by slightly different rules than is a normal formula. In a normal formula, if you give a reference to a range of cells as an argument to a function or operator that normally operates on single values, Excel tries to select just one value from one cell of the reference to operate on. It makes this selection as described in "Operators" in Chapter 4 of the *User's Guide*. In an array formula, however, Excel performs the function or operation on each single value within the array to produce an array as a result.

Relative references within an array formula are always considered to be relative to the upper-left corner cell of the range of cells containing the formula.

Array Arguments and Results

In an array formula, if any argument to a function or operator that is supposed to be a single value is in fact an array, then all such arguments to that function or operator should be arrays of the same size. This allows Excel to perform the function or operation on corresponding successive elements within the arrays.

Expanding array arguments

If these arguments are not all arrays of the same size, Excel automatically expands them so that they are the same size.

For example, in the formula =SUM({1,2,3}*4), one argument to the multiplication operator is a 1 by 3 array and one is a single value. In evaluating this formula, Excel automatically expands the second argument to a 1 by 3 array and evaluates the formula as =SUM({1,2,3}*{4,4,4}), the value of which is 24.

In the above example, Excel considers both arguments for expansion. Each must have the same number of rows as the argument with the greatest number of rows and the same number of columns as the argument with the greatest number of columns. Since the first argument contains the greatest number of both rows and columns, it does not get changed.

If an argument is a single value, or a 1 by 1 array, it is expanded by repeating its single value as often as necessary to produce the desired array. If an argument is a one-row or one-column array, it is expanded by repeating its one row or column of values. Otherwise, if an argument is an array with multiple rows and columns, Excel expands it to the proper size using the error value #N/A for the additional values.

Examples

The following table shows how array arguments, as described above, are expanded.

In this formula	Excel expands the arguments like this
1+{1,2;3,4}	{1,1;1,1}+{1,2;3,4}
{1,2,3}*{2;3}	{1,2,3;1,2,3}*{2,2,2;3,3,3}
{1,2}+{1,2,3;4,5,6}	{1,2,#N/A;1,2,#N/A}+{1,2,3;4,5,6}
{1,2;3,4;5,6} +{1,2,3;4,5,6}	{1,2,#N/A;3,4,#N/A;5,6,#N/A} +{1,2,3;4,5,6;#N/A,#N/A,#N/A}
MOD({5,6,7},4)	MOD({5,6,7},{4,4,4})
IF({"A";"B";"E"}={"C"; "B";"A"},TRUE,FALSE)	IF({"A";"B";"E"}={"C";"B";"A"}, {TRUE;TRUE;TRUE},{FALSE;FALSE;FALSE})

More on expanding array arguments

As stated earlier: in an array formula, if any argument to a function or operator that is supposed to be a single value is in fact an array, then all such arguments to the function or operator should be arrays of the same size. If they are not, Excel automatically expands them so that they are. However, Excel never expands an argument that is supposed to be an array.

For example, the MIRR function has three arguments, *values*, *safe*, and *risk*. *Safe* and *risk* are supposed to be single values. *Values* is supposed to be an array. If either *safe* or *risk* is in fact an array rather than a single value, both arguments are expanded to arrays of the same size. But those arguments are not expanded to the same size as *values*, which is supposed to be an array, nor is *values* ever expanded to be the same size as *safe* and *risk*.

For example, in the formula:

MIRR({−5000;1000;1500;2000;−3000;2500;5000},
{0.12,0.13,0.14},0.17)

Excel considers the second and third arguments for expansion. Since the second argument is already a 1 by 3 array, only the third argument actually gets expanded. The *risk* argument is expanded only to a 1 by 3 array (to correspond to the size of the *safe* argument). It is not expanded to a 7 by 3 array to match the larger dimension of the 7 by 1 *values* argument. The expanded formula:

MIRR({−5000;1000;1500;2000;−3000;2500;5000},
{0.12,0.13,0.14},{0.17,0.17,0.17})

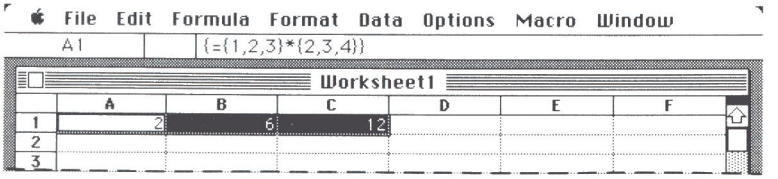
then produces the 1 by 3 array {0.1519,0.1537,0.1553}.

For an explanation of which worksheet functions accept arrays for arguments, see "Arguments to Functions" in Chapter 3, "Worksheet Function Directory."

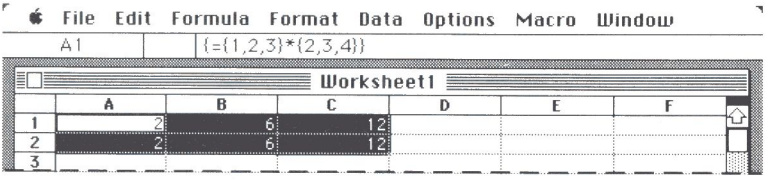
Producing arrays as results

When you enter an array formula into a range of cells, the formula should, ideally, produce an array of the same size as the selected range of cells; this allows each value in the array to be placed in one cell of the range. If the array is smaller than the range of cells, the array will be expanded, as described above, before its values are placed in the cells. If the array is larger than the range, the excess values will not be placed in cells at all.

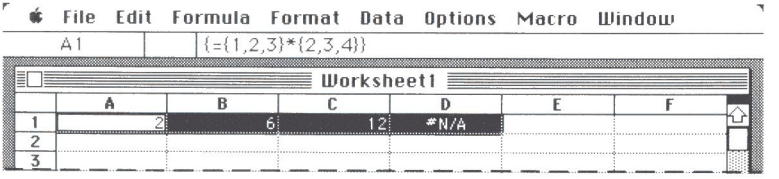
For example, if the array formula ={1,2,3}*{2,3,4} is entered into the 1 by 3 range of cells A1:C1, Excel produces the 1 by 3 array {2,6,12}.



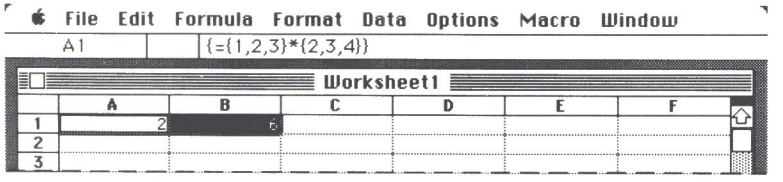
If that formula were entered into the 2 by 3 range of cells A1:C2, Excel would expand the result to {2,6,12;2,6,12} before placing it in the cells.



If the formula were entered into the 1 by 4 range of cells A1:D1, Excel would expand the result to {2,6,12,#N/A}.



If the formula were entered into the 1 by 2 range of cells A1:B1, Excel would trim the result to {2,6}.



Examples

The following table shows how arrays are produced as results.

For this formula	Excel produces this result
{1,2,3}+{4,5,6}	{5,7,9}
SQRT({1,4,9})	{1,2,3}
{1,2,3}*{2;3}	{2,4,6;3,6,9}
{1,2}+{1,2,3;4,5,6}	{2,4,#N/A;5,7,#N/A}
{1,2;3,4;5,6}+{1,2,3;4,5,6}	{2,4,#N/A;7,9,#N/A;#N/A,#N/A,#N/A}
MOD({5,6,7},4)	{1,2,3}
IF({"A";"B";"E"} ={"C";"B";"A"},TRUE,FALSE)	{FALSE;TRUE;FALSE}

Array Functions

Excel has several functions which always return arrays of values. The size of the returned array varies depending on the particular function being used.

Some functions, like TREND, return an array of the same size as their arguments. For example, the array formula =TREND(B2:G2) returns a 1 by 6 array of values because the reference B2:G2 is a 1 by 6 array. Therefore, this array formula should be entered into a 1 by 6 range of cells such as B3:G3.

Another function, TRANSPOSE, returns an array whose size is the transpose of its argument. For example, the array formula =TRANSPOSE({1,2,3;4,5,6}) takes the 2 by 3 array {1,2,3;4,5,6} and returns the 3 by 2 array {1,4;2,5;3,6}.

Still other functions always return an array of the same size, regardless of the size of their arguments. For example, the LINEST function always returns a 1 by 2 array. If you want to enter this array into a 2 by 1 range of cells, you must use the TRANSPOSE function.

The behavior of each array function is described in Chapter 3, "Worksheet Function Directory."

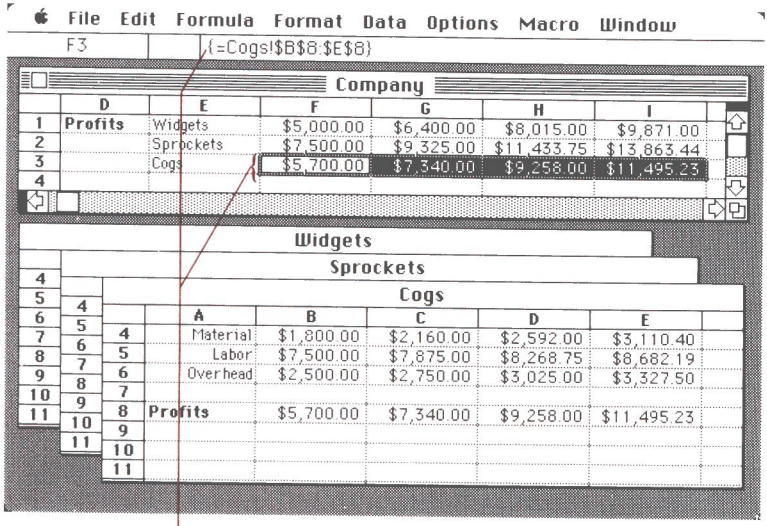
Arrays and External References

Array formulas involving external references behave just like other array formulas. However, they deserve special mention because so many array formulas involve external references. For a complete description of how external references behave, see "Linking Documents" in Chapter 8 of the *User's Guide*.

Linking values between worksheets

One of the most common uses for external references is to link a range of cells on one worksheet to a range of cells on another.

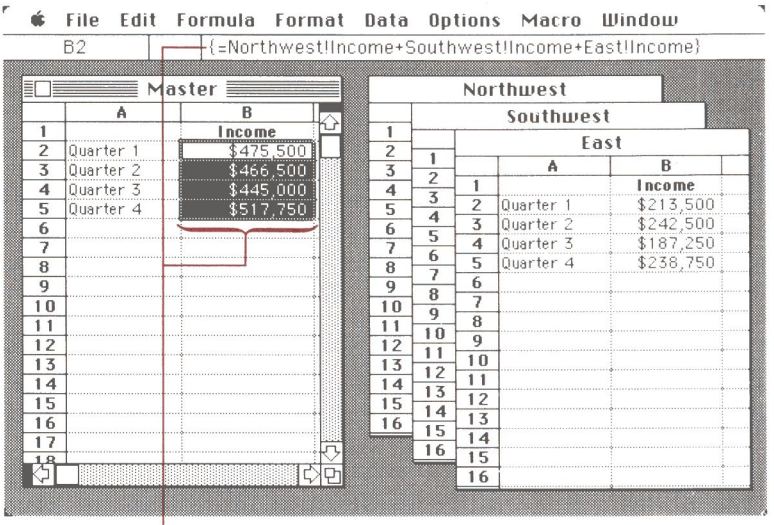
For example, if the Cogs worksheet contains values in cells B8:E8 that you want to use on the Company worksheet in cells F3:I3, you can enter in those cells the array formula =Cogs!$B$8:$E$8. Then, when you change the original values on Cogs, Excel recalculates the array formula on Company to produce the new values on that worksheet.



The array formula in F3:I3 copies the values from B8:E8 on the Cogs worksheet.

Consolidating values onto a master worksheet

Another common use for external references is to consolidate values from ranges of cells on different worksheets onto one master worksheet, as in the example given earlier involving the array formula =Northwest!Income+Southwest!Income+East!Income.



The array formula consolidates the income values from the Northwest, Southwest, and East worksheets.

Arrays and Charting

In a chart, all series formulas contain arrays. Specifically, the arguments, *categories* and *values*, to the SERIES function must each be an array of values. For a complete description of series formulas and the SERIES function, see "Using Series Formulas" in Chapter 7 of the *User's Guide*. In that section, these arguments are always references to cells that contain an array of values.

In most cases, you want to store your values in worksheet cells. Then, in your series formulas in charts and in your formulas on worksheets, you can refer to the cells containing those values. When Excel creates series formulas in a chart for you, either automatically or with the Copy and Paste commands, it always uses references.

However, as with array formulas on worksheets, it is possible to use an array constant instead of a range of cells containing an array. When you enter a series formula yourself by typing in the formula bar, you can type in an array constant.

For example, suppose you have a worksheet named Budget, on which cells A1:C1 contain the text values Pens, Clips, and Tape; and cells A2:C2 contain the numbers 10, 20, and 30. The series formula:

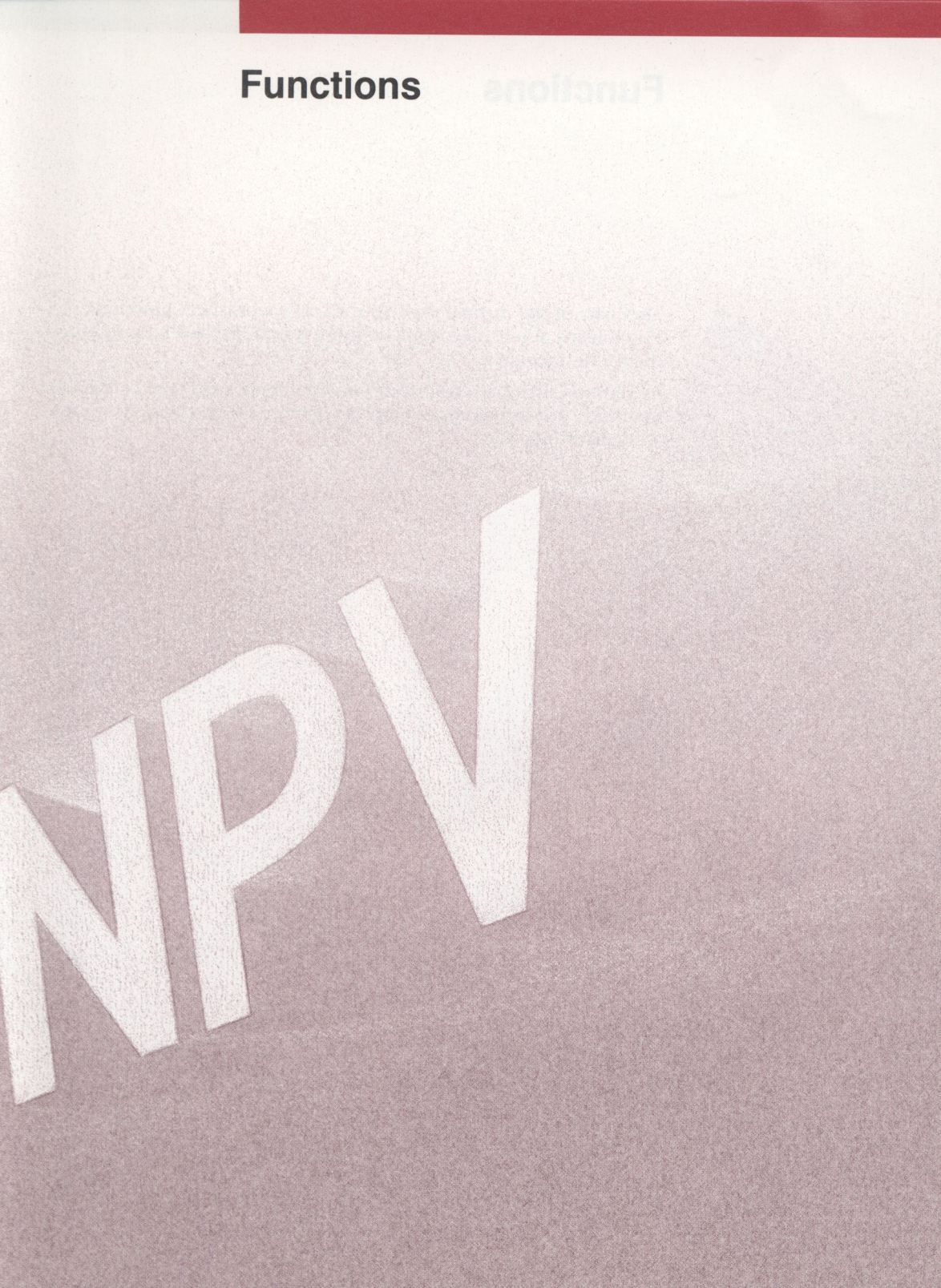
SERIES("Actual",Budget!$A$1:$C$1,Budget!$A$2:$C$2,1)

is equivalent to the series formula:

SERIES("Actual",{"Pens","Clips","Tape"},{10,20,30},1).

In the first series formula, if you select $A$2:$C$2 in the formula bar and choose the Calculate Now command, the reference is replaced with the array constant {10,20,30}, exactly as happens if the same reference appeared in an array formula on a worksheet.

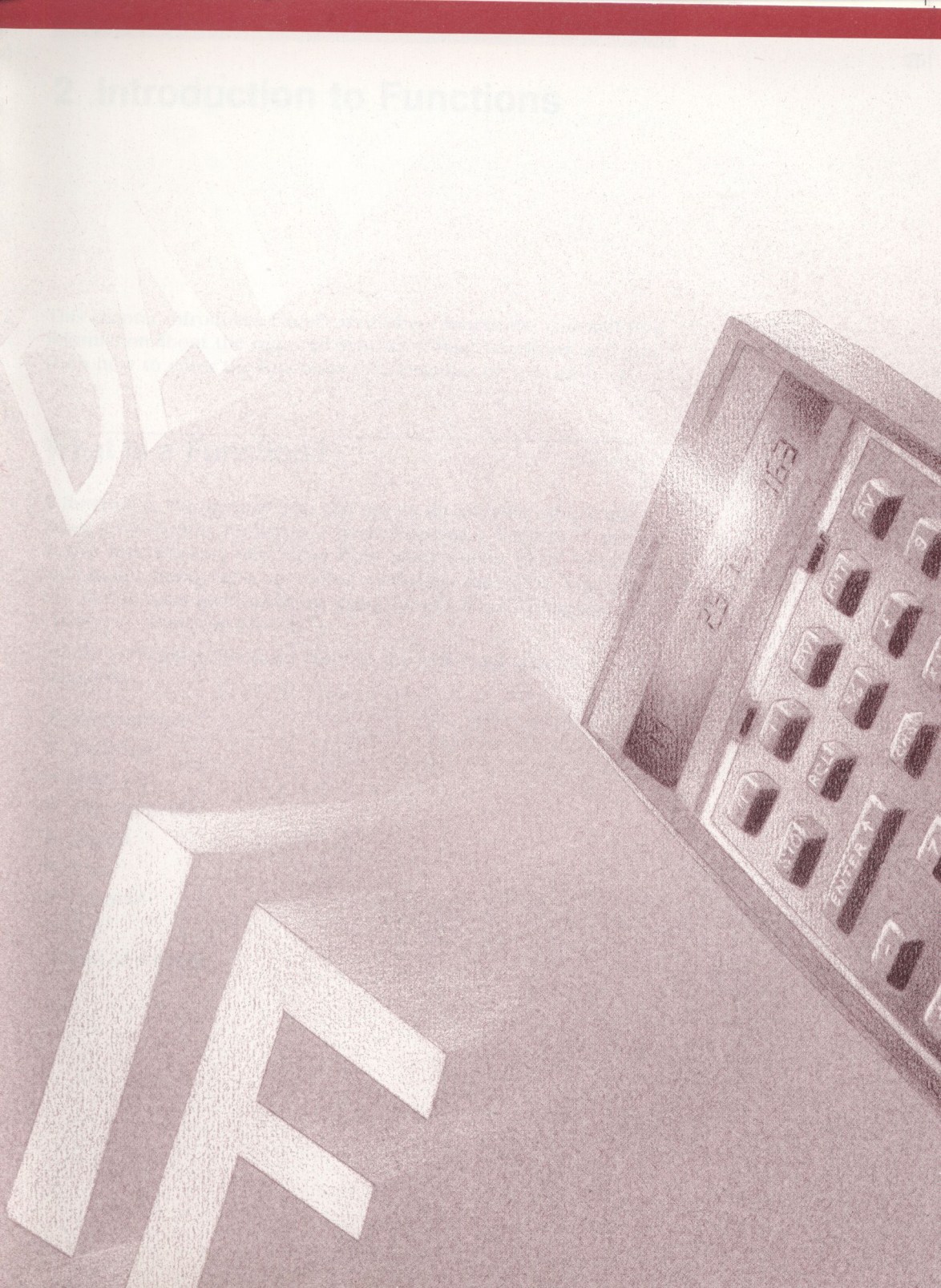
Functions



Functions

This part of the manual describes Excel's worksheet functions. You'll learn about the syntax of these functions, and how to enter them into formulas.

In addition, this part describes the arguments expected by Excel's functions, and provides a complete listing and description of each worksheet function.


DA
IF
ENTER

2 Introduction to Functions

This chapter introduces Excel's worksheet functions. You will find information about the required syntax of these functions, and you'll learn how to enter the functions into formulas on a worksheet.

What Is a Function?

A function is an operator you can use to do a variety of calculations. A function is similar to a symbol operator, such as + or −, in that it produces a new value from other values. Excel has many built-in functions; these are called worksheet functions. You can also define your own functions using Excel's Macro Language (see Chapter 6, "Function Macros").

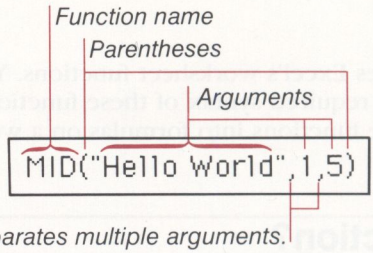
Excel's worksheet functions fall into the following general categories:

- Mathematical
- Statistical
- Database
- Trigonometric
- Logical
- Text
- Financial
- Date
- Special-purpose

Syntax of Functions

Parts of a function

Each function consists of the function name, a set of parentheses, and arguments, which are the values the function uses to produce a new value.



The parentheses are very important. They tell Excel where the arguments begin and end. When you paste a function using the Paste Function command from the Edit menu, Excel provides the parentheses and positions the insertion point between them. All you have to do is type the arguments and the commas.

If you type the function instead of pasting it, remember to type the parentheses. Do not leave any space before the left parenthesis, and be sure to finish with a right parenthesis. If you don't, you will see an alert box that says "Error in formula."

Entering Functions into Formulas

Entering a function into a formula

You can enter a function in any formula you build on a worksheet. Use the Paste Function command from the Edit menu to paste a function into a formula. Or, type the function. You can use uppercase or lowercase letters or a combination of both; Microsoft Excel does not distinguish between cases.

Here are some examples of different functions and what they do:

This function	Performs this action
SUM(1,2,3)	Uses the SUM function to add 1, 2, and 3.
AVERAGE(1,2,3)	Uses the AVERAGE function to calculate the average of 1, 2, and 3.
SIN(0.53)	Uses the SIN function to calculate the sine of an angle of 0.53 radians.
LEN("Phil Smith")	Uses the LEN function to count the number of characters in the text Phil Smith.

The arguments to a function can be constants, as in the above examples, or they may be formulas. These formulas, themselves, may contain other functions. In the example below, the formula Score/10 is an argument to the INT function. This function is in turn an argument to the MID function.

```
MID("FFFFFFDCBAA",INT(Score/10),1)
```

When an argument to a function is itself a function, it is said to be nested. In Excel, you can nest functions up to seven levels in a formula.

3 Worksheet Function Directory

This chapter is divided into three sections: a discussion of the arguments expected by Excel functions, a brief listing of the available worksheet functions arranged by subject category, and a detailed alphabetical directory of these functions.

Arguments to Functions

A function can have from 0 to 14 arguments. These arguments can be numbers, text, logical values, arrays, or references.

Most arguments in Excel functions are expected to be of a certain type. The argument you give should be either a value of that type or a formula that evaluates to a value of that type.

For example, the SUM function, which computes the sum of its arguments, expects numbers as arguments. The arguments to this function can be given in a number of ways. You can give constants as arguments, as in SUM(100,200,300). Or, you can use the form SUM(Rent,Tax,Insurance), if those names evaluate to numbers. You can also use the form SUM(A1,B1,C1), if those cells contain numbers.

In the worksheet function directory that follows, each argument is described by one of the following:

- Its type (i.e., number, numbers, text, logical, logicals, reference, value, values, array, or vector)
- A descriptive term, such as "criteria" or "rate," that is explained in the text
- A combination of the type and the descriptive term, such as "serial-number" or "X-array"

The different types of arguments are described in the following table.

This argument	Can be any of these
Number	Anything that produces a number: a number or numeric formula, or a reference or reference formula that refers to a cell whose value is a number
Numbers	Anything that produces one or more numbers: a number as defined above, an array or array formula of numbers, or a reference or reference formula that refers to cells whose values are numbers
Text	Anything that produces text: text or a text formula, or a reference or reference formula that refers to a cell whose value is text
Logical	Anything that produces a logical value: a logical formula, or a reference or reference formula that refers to a cell whose value is a logical value
Logicals	Anything that produces one or more logical values: a logical value as defined above, an array or array formula of logical values, or a reference or reference formula that refers to cells whose values are logical values
Ref(erence)	Anything that produces a reference: a reference or reference formula
Value	Anything that produces a value: any formula or value
Values	Anything that produces one or more values: a value as defined above, an array or array formula, or a reference or reference formula
Array	Anything that produces an array: an array or array formula, or a reference or reference formula that refers to a rectangular range of cells
Vector	Anything that produces a one-dimensional array: a one-dimensional array or array formula, or a reference or reference formula that refers to a one-dimensional range of cells

Note Text arguments must be enclosed in quotation marks. If the text itself contains quotation marks, you need to include two sets of quotation marks in order to have Excel display the text accurately. For example, to have Excel give the length of this text — in the "good" old days — you need to enter: LEN("in the ""good"" old days").

Giving arrays as arguments

In the function descriptions that follow, arguments that are supposed to be single values are denoted by *number*, *text*, *logical*, or *value*, or they are described as such in the text. Arguments that are not supposed to be single values are denoted by *numbers*, *logicals*, *ref*, *values*, *array*, or *vector*, or they are described as such in the text.

In an array formula, if you give an array as an argument to a function, such as ABS, that expects a single value argument and produces a single value as a result, Excel performs the function on each successive value within the array to produce an array as a result. If you give more than one such argument as an array, Excel performs the function on corresponding successive values within the arrays, again to produce an array as a result. For example, the FV function has five arguments that are supposed to be single values, and it produces a single value as a result. Another example is the MIRR function. Even though it has one argument that is supposed to be an array, it has two arguments that are supposed to be single values, and it produces a single value as a result. Most of Excel's worksheet functions operate like this; that is, they have either one or more arguments that are supposed to be single values, and they produce a single value as a result.

If a function has no arguments, or if none of the arguments are supposed to be single values, or if the function produces an array or reference as a result, then it does not behave as described above, but rather behaves exactly as stated in the function description. An example is the SUM function, all of whose arguments can be arrays or references. Another example is the TREND function, which always produces an array as a result.

The following functions have no arguments, have no arguments that are supposed to be single values, and/or produce arrays or references as results:

Function	Characteristic(s)
AND	Has no arguments that are supposed to be single values
AREAS	Has no arguments that are supposed to be single values
AVERAGE	Has no arguments that are supposed to be single values
COLUMN	Has no arguments that are supposed to be single values; produces an array or reference as a result
COLUMNS	Has no arguments that are supposed to be single values
COUNT	Has no arguments that are supposed to be single values
FALSE	Has no arguments
GROWTH	Has no arguments that are supposed to be single values; produces an array or reference as a result
INDEX	Produces an array or reference as a result
ISREF	Has no arguments that are supposed to be single values
LINEST	Has no arguments that are supposed to be single values; produces an array or reference as a result
LOGEST	Has no arguments that are supposed to be single values; produces an array or reference as a result
MAX	Has no arguments that are supposed to be single values
MIN	Has no arguments that are supposed to be single values
NA	Has no arguments
NOW	Has no arguments

Function	Characteristic(s)
OR	Has no arguments that are supposed to be single values
PI	Has no arguments
RAND	Has no arguments
ROW	Has no arguments that are supposed to be single values; produces an array or reference as a result
ROWS	Has no arguments that are supposed to be single values
STDEV	Has no arguments that are supposed to be single values
SUM	Has no arguments that are supposed to be single values
TRANSPOSE	Has no arguments that are supposed to be single values; produces an array or reference as a result
TREND	Has no arguments that are supposed to be single values; produces an array or reference as a result
TRUE	Has no arguments
VAR	Has no arguments that are supposed to be single values

Functions by Subject Category

The detailed directory of functions in this chapter is presented in alphabetical order for quick reference. However, to help you readily see what functions are available in Excel, they are also presented below, grouped by subject category.

Mathematical Functions

ABS(*number*)
Absolute value of *number*

EXP(*number*)
e to the power *number*

INT(*number*)
Integer part of *number*

LN(*number*)
Logarithm, base e, of *number*

LOG10(*number*)
Logarithm, base 10, of *number*

MOD(*number,divisor-number*)
Remainder of *number* divided by *divisor-number*

PI( )
Value of π

RAND( )
Random number between 0 and 1

ROUND(*number,number-of-digits*)
Number rounded to *number-of-digits*

SIGN(*number*)
Sign of *number*

SQRT(*number*)
Square root of *number*

Statistical functions

Statistical Functions

AVERAGE(*numbers-1,numbers-2,...*)
Average of numbers in *numbers*

COUNT(*numbers-1,numbers-2,...*)
Count of numbers in *numbers*

GROWTH(*Y-array,X-array,x-array*)
Values on exponential trend y = b*m^x

LINEST(*Y-array,X-array*)
Parameters of linear trend y = m*x + b

LOGEST(*Y-array,X-array*)
Parameters of exponential trend y = b*m^x

MAX(*numbers-1,numbers-2,...*)
Maximum number in *numbers*

MIN(*numbers-1,numbers-2,...*)
Minimum number in *numbers*

STDEV(*numbers-1,numbers-2,...*)
Standard deviation of numbers in *numbers*

SUM(*numbers-1,numbers-2,...*)
Sum of numbers in *numbers*

TREND(*Y-array,X-array,x-array*)
Values on linear trend y = m*x + b

VAR(*numbers-1,numbers-2,...*)
Variance of numbers in *numbers*

Database Functions

DAVERAGE(*database,field-name,criteria*)
DAVERAGE(*database,field-index,criteria*)
Average of numbers in specified *field* of *database* records matching *criteria*

DCOUNT(*database,field-name,criteria*)
DCOUNT(*database,field-index,criteria*)
Count of numbers in specified *field* of *database* records matching *criteria*

DMAX(*database,field-name,criteria*)
DMAX(*database,field-index,criteria*)
Maximum of numbers in specified *field* of *database* records matching *criteria*

DMIN(*database,field-name,criteria*)
DMIN(*database,field-index,criteria*)
Minimum of numbers in specified *field* of *database* records matching *criteria*

DSTDEV(*database,field-name,criteria*)
DSTDEV(*database,field-index,criteria*)
Standard deviation of numbers in specified *field* of *database* records matching *criteria*

DSUM(*database,field-name,criteria*)
DSUM(*database,field-index,criteria*)
Sum of numbers in specified *field* of *database* records matching *criteria*

DVAR(*database,field-name,criteria*)
DVAR(*database,field-index,criteria*)
Variance of numbers in specified *field* of *database* records matching *criteria*

Trigonometric Functions

ACOS(*number*)
Arccosine of *number*

ASIN(*number*)
Arcsine of *number*

ATAN(*number*)
Arctangent of *number*

ATAN2(*x-number,y-number*)
Arctangent of point (*x-number,y-number*)

COS(*number*)
Cosine of *number*

SIN(*number*)
Sine of *number*

TAN(*number*)
Tangent of *number*

Logical functions

Logical Functions

AND(*logicals-1,logicals-2,...*)
TRUE if all logical values in *logicals* are TRUE; otherwise, FALSE

CHOOSE(*index,value-1,value-2,...*)
Uses *index* to select a value from *values*

FALSE( )
Logical value FALSE

IF(*logical,value-if-true,value-if-false*)
Value-if-true if *logical* is TRUE; *value-if-false* if *logical* is FALSE

ISERROR(*value*)
TRUE if *value* is any error value

ISNA(*value*)
True if *value* is the error value #N/A

ISREF(*value*)
TRUE if *value* is a reference

NOT(*logical*)
TRUE if *logical* is FALSE; FALSE if *logical* is TRUE

OR(*logicals-1,logicals-2,...*)
TRUE if any logical value in *logicals* is TRUE; otherwise, FALSE

TRUE( )
Logical value TRUE

Text functions

Text Functions

DOLLAR(*number,number-of-digits*)
Rounds *number* and gives text in currency format

FIXED(*number,number-of-digits*)
Rounds *number* and gives text in fixed format

LEN(*text*)
Length of *text*

MID(*text,start-position,number-of-characters*)
Extracts *number-of-characters* from *text*

REPT(*text,number-of-times*)
Repeats *text number-of-times*

SEARCH(*search-text,text,start-number*)
Searches for *search-text* in *text*

TEXT(*number,format-text*)
Converts *number* to text using *format-text*

VALUE(*text*)
Converts *text* to a number

Financial Functions

Financial functions

FV(*rate,nper,pmt,pv,type*)
Future value of investment

IRR(*values,guess*)
Internal rate of return of *values*

MIRR(*values,safe,risk*)
Modified internal rate of return of *values*

NPER(*rate,pmt,pv,fv,type*)
Number of payments of investment

NPV(*rate,values-1,values-2,...*)
Net present value of *values*

PMT(*rate,nper,pv,fv,type*)
Periodic payment of investment

PV(*rate,nper,pmt,fv,type*)
Present value of investment

RATE(*nper,pmt,pv,fv,type,guess*)
Rate returned on investment

Date Functions

Date functions

DATE(*year,month,day*)
Serial number of specified date

DAY(*serial-number*)
Converts *serial-number* to a day of the month

HOUR(*serial-number*)
Converts *serial-number* to an hour of the day

MINUTE(*serial-number*)
Converts *serial-number* to a minute

MONTH(*serial-number*)
Converts *serial-number* to a month of the year

NOW( )
Serial-number of current date and time

SECOND(*serial-number*)
Converts *serial-number* to a second

TIME (*hour,minute,second*)
Serial number of specified time

WEEKDAY(*serial-number*)
Converts *serial-number* to a day of the week

YEAR(*serial-number*)
Converts *serial-number* to a year

Special-purpose functions

Special-Purpose Functions

AREAS(*ref*)
Number of areas in *ref*

COLUMN(*ref*)
Column numbers in *ref*

COLUMNS(*array*)
Number of columns in *array*

HLOOKUP(*lookup-value,compare-array,index-number*)
Value in a table selected by *lookup-value*

INDEX(*ref,row,column,area*)
INDEX(*array,row,column*)
Reference in *ref* or value in *array* selected by index values

LOOKUP(*lookup-value,compare-vector,result-vector*)
LOOKUP(*lookup-value,compare-array*)
Value in a table selected by *lookup-value*

MATCH(*lookup-value,compare-vector, type*)
Number of a value selected by *lookup-value*

NA( )
Error value #N/A

ROW(*ref*)
Row numbers in *ref*

ROWS(*array*)
Number of rows in *array*

TRANSPOSE(*array*)
Transpose of *array*

TYPE(*value*)
Type of value

VLOOKUP(*lookup-value,compare-array,index-number*)
Value in a table selected by *lookup-value*

ABS(number)

ABS

The ABS function gives the absolute value of *number*.

Examples

ABS(2) *equals* ABS(−2) *equals* 2

ACOS(number)

ACOS

The ACOS function gives the arccosine of *number*. The arccosine is the angle in radians whose cosine is *number*. *Number* must be in the range −1 to 1. The angle will be in the range 0 to π.

Examples

ACOS(−0.5) *equals* 2.094 (2π/3 radians)

ACOS(−0.5)*180/PI() *equals* 120 (degrees)

AND(logicals-1,logicals-2,...)

AND

The AND function gives the logical value TRUE if all the logical values in the list of arguments are TRUE. If any of the logical values is FALSE, AND gives the logical value FALSE.

AND can have from 1 to 14 arguments. The arguments should be logical values, or arrays or references that contain logical values. If an array or reference argument contains text or number values, those values are ignored.

Examples

AND(2+2=4,2+3=5) *equals* TRUE

AND(2+2=5,2+3=5) *equals* FALSE

AREAS

AREAS(ref)

The AREAS function gives the number of areas in *ref*. An area is a range of cells or a single cell. *Ref* can be a reference to multiple areas.

Examples

AREAS((B2:D4,E5,F6:I9)) *equals* 3

AREAS(B2:D4) *equals* 1

ASIN

ASIN(number)

The ASIN function gives the arcsine of *number*. The arcsine is the angle in radians whose sine is *number*. *Number* must be in the range −1 to 1. The angle will be in the range $-\pi/2$ to $\pi/2$.

Examples

ASIN(−0.5) *equals* −0.524 ($-\pi/6$ radians)

ASIN(−0.5)*180/PI() *equals* −30 (degrees)

ATAN

ATAN(number)

The ATAN function gives the arctangent of *number*. The arctangent is the angle in radians whose tangent is *number*. The angle will be in the range $-\pi/2$ to $\pi/2$.

Examples

ATAN(1) *equals* 0.785 ($\pi/4$ radians)

ATAN(1)*180/PI() *equals* 45 (degrees)

ATAN2(x-number,y-number)

ATAN2

The ATAN2 function gives the arctangent of *x-number* and *y-number*. The arctangent is the angle in radians determined by the point whose x and y coordinates are *x-number* and *y-number*. The angle will be in the range $-\pi$ to π, excluding $-\pi$.

If both *x-number* and *y-number* are 0, ATAN2 gives the error value #DIV/0!.

Examples

ATAN2(1,1) *equals* 0.785 ($\pi/4$ radians)

ATAN2(−1,−1) *equals* −2.356 ($-3\pi/4$ radians)

ATAN2(−1,−1)*180/PI( ) *equals* −135 (degrees)

AVERAGE(numbers-1,numbers-2,...)

AVERAGE

The AVERAGE function gives the average of the numbers in the list of arguments.

AVERAGE can have from 1 to 14 arguments. The arguments should be numbers, or arrays or references that contain numbers. If an array or reference argument contains text, logical, or blank values, those values are ignored.

Examples

If A1:A5 contains the numbers 10, 7, 9, 27, and 2, then:

AVERAGE(A1:A5) *equals* 11

AVERAGE(A1:A5,5) *equals* 10

AVERAGE(A1:A5) *equals* SUM(A1:A5)/COUNT(A1:A5)

CHOOSE

CHOOSE(index,value-1,value-2,...)

The CHOOSE function uses *index* to select a value from the following arguments in the list. If *index* is 1, CHOOSE gives *value-1*; if *index* is 2, CHOOSE gives *value-2*; and so on.

If *index* is less than 1 or greater than the number of the last value in the list, CHOOSE gives the error value #VALUE!.

Example

CHOOSE(2,"1st","2nd","3rd","Finished") *equals* "2nd"

COLUMN

COLUMN(ref)

The COLUMN function gives the column number of *ref*, if *ref* is the reference of a single cell. If *ref* is a range of cells, COLUMN gives the column numbers of *ref* as a horizontal array. If *ref* is omitted, it is assumed to be the reference of the cell(s) in which the COLUMN function appears.

Ref cannot be a reference to multiple areas.

Examples

COLUMN(A3) *equals* 1

COLUMN(A3:C5) *equals* {1,2,3}

If COLUMN is entered in C5, then:

COLUMN( ) *equals* COLUMN(C5) *equals* 3

COLUMNS

COLUMNS(array)

The COLUMNS function gives the number of columns in *array*.

Examples

COLUMNS(A1:C4) *equals* 3

COLUMNS({1,2,3;4,5,6}) *equals* 3

COS(number)

COS

The COS function gives the cosine of *number,* where *number* is an angle in radians.

Examples

COS(1.047) *equals* 0.5

COS(60*PI()/180) *equals* 0.5

COUNT(numbers-1,numbers-2,...)

COUNT

The COUNT function gives the count of the numbers in the list of arguments.

COUNT can have from 1 to 14 arguments. The arguments should be numbers, or arrays or references that contain numbers. If an array or reference argument contains text, logical, error, or blank values, those values are ignored.

Examples

If A1:A5 contains the numbers 10, 7, 9, 27, and 2, then:

COUNT(A1:A5) *equals* 5

COUNT(A1:A5,5) *equals* 6

DATE(year,month,day)

DATE

The DATE function gives the serial number of the date specified by *year*, *month*, and *day*. For an explanation of dates and serial numbers, see "Entering a Date or Time" in Chapter 4 of the *User's Guide*.

The serial number is an integer in the range 0 to 49710, representing dates from January 1, 1904 to February 6, 2040. The arguments *year*, *month*, and *day* should make up a valid date in this range. That is, *year* should be between 1904 and 2040; *month* should be between 1 and 12; and *day* should be between 1 and 31. Note that *year* can also be between 4 and 140, representing the years 1904 to 2040.

It is not necessary to enter a formula like =DATE(1985,4,15) in a cell, because it's easier to enter the date constant 4/15/1985 or 15-April-1985 in the cell to produce the same serial number. Likewise, it is not necessary to use a formula like =DATE(1985,4,15) within a larger formula, because it's easier to use the text "4/15/1985" or "15-April-1985" in the formula; Excel automatically converts the text into the serial number, as necessary. However, if you want *year*, *month*, or *day* to be formulas rather than constants, you must use the DATE function.

Examples

DATE(1985,CHOOSE(1,3,6,9,12),1)
equals DATE(1985,3,1) *equals* 29645

DATE(1985,4,15)−DATE(1985,1,1)
equals "4/15/1985"−"1/1/1985" *equals* 104

DAVERAGE

DAVERAGE(database,field-name,criteria)
DAVERAGE(database,field-index,criteria)

The DAVERAGE function gives the average of the numbers in a particular field of a database for all records that satisfy a set of criteria.

The database is specified by *database*, which must be the reference of a valid database range. If you have specified a database with the Set Database command, you can use the name Database, defined by that command.

The particular field within the database is specified by the text argument *field-name*, which must be a valid field name within *database*. Or, you can use *field-index*; a 1 specifies the first field in the database, a 2 specifies the second field, and so on.

If you omit *field-name* or *field-index*, DAVERAGE gives the average of all records in *database* that satisfy *criteria*, rather than the average of only the numbers in the specified *field*.

The criteria are specified by *criteria*, which must be the reference of a valid criteria range. If you have specified a criteria range with the Set Criteria command, you can use the name Criteria, defined by that command.

Note *database* and *criteria* cannot be external references. A database statistical function and the database and criteria ranges it uses must be on the same worksheet.

For more information, see "Creating a Database" and "Creating Criteria" in Chapter 6 of the *User's Guide*.

DAVERAGE is the database equivalent of the statistical function AVERAGE. Seven of the statistical functions (AVERAGE, COUNT, MAX, MIN, STDEV, SUM, and VAR) have database equivalents (DAVERAGE, DCOUNT, DMAX, DMIN, DSTDEV, DSUM, and DVAR). See the descriptions of the statistical functions for explanations of the operation of the equivalent database functions.

For example, DAVERAGE(Database,"Amount",Criteria) averages all the numbers in the field called "Amount" from the records in the range named Database that match the criteria in the range named Criteria. DAVERAGE(Database,,Criteria) gives the average of all the records in the range named Database that match the criteria in the range named Criteria.

DAY(serial-number)
MONTH(serial-number)
YEAR(serial-number)
WEEKDAY(serial-number)

DAY
MONTH
YEAR
WEEKDAY

The DAY function gives the number of the day in *serial-number* as an integer in the range 1 to 31. The MONTH function gives the number of the month as an integer in the range 1 to 12. The YEAR function gives the number of the year as an integer in the range 1904 to 2040. The WEEKDAY function gives the number of the weekday as an integer in the range 1 to 7. For an explanation of dates, times, and serial numbers, see "Entering a Date or Time" in Chapter 4 of the *User's Guide*.

Serial-number should be in the range 0 to 49710 to represent a valid date. If *serial-number* is not an integer, only the integer portion is used, since dates are represented as integers. You may give *serial-number* as text, such as "4/15/1985" or "15-April-1985", instead of as a number; Excel automatically converts the text into the serial number.

Examples

DAY(DATE(1985,4,15)) *equals* 15

MONTH(DATE(1985,4,15)) *equals* 4

YEAR(29690) *equals* 1985

WEEKDAY("4/15/1985") *equals* 2

DCOUNT

DCOUNT(database,field-name,criteria)
DCOUNT(database,field-index,criteria)

DCOUNT is the database equivalent of the statistical function COUNT. The DCOUNT function gives the count of the numbers in a particular field of a database for all records that satisfy a set of criteria. If you omit the *field* argument, DCOUNT counts all the records that match the criteria, even if the fields in those records do not contain numbers.

For example, DCOUNT(Database,"Amount",Criteria) gives the count of all the numbers in the field called "Amount" from the records in the range named Criteria.

For a complete description of the arguments in database functions, see DAVERAGE.

See also COUNT.

DMAX

DMAX(database,field-name,criteria)
DMAX(database,field-index,criteria)

The DMAX function gives the largest number in a particular field of a database for all records that satisfy a set of criteria. DMAX is the database equivalent of the statistical function MAX.

For a complete description of the arguments in database functions, see DAVERAGE.

See also MAX.

DMIN

DMIN(database,field-name,criteria)
DMIN(database,field-index,criteria)

The DMIN function gives the smallest number in a particular field of a database for all records that satisfy a set of criteria. DMIN is the database equivalent of the statistical function MIN.

For a complete description of the arguments in database functions, see DAVERAGE.

See also MIN.

DOLLAR(number,number-of-digits)

DOLLAR

The DOLLAR function rounds *number* to *number-of-digits*, formats it in currency format, and gives the result in the form of text.

If *number-of-digits* is negative, *number* is rounded to the left of the decimal point. If you omit *number-of-digits*, it is assumed to be 2.

Examples

DOLLAR(1234.567,2) *equals* "$1,234.57"

DOLLAR(−1234.567,−2) *equals* "($1,200)"

DOLLAR(−0.123,4) *equals* "($0.1230)"

DOLLAR(1234.567) *equals* "$1,234.57"

DOLLAR(−0.123) *equals* "($0.12)"

DSTDEV(database,field-name,criteria)
DSTDEV(database,field-index,criteria)

DSTDEV

The DSTDEV function gives the sample standard deviation of the numbers in a particular field of a database for all records that satisfy a set of criteria. DSTDEV is the database equivalent of the statistical function STDEV.

For a complete description of the arguments in database functions, see DAVERAGE.

See also STDEV.

DSUM(database,field-name,criteria)
DSUM(database,field-index,criteria)

DSUM

The DSUM function gives the sum of the numbers in a particular field of a database for all records that satisfy a set of criteria. DSUM is the database equivalent of the statistical function SUM.

For a complete description of the arguments in database functions, see DAVERAGE.

See also SUM.

DVAR

DVAR(database,field-name,criteria)
DVAR(database,field-index,criteria)

The DVAR function gives the sample variance of the numbers in a particular field of a database for all records that satisfy a set of criteria. DVAR is the database equivalent of the statistical function VAR.

For a complete description of the arguments in database functions, see DAVERAGE.

See also VAR.

EXP

EXP(number)

The EXP function gives e raised to the power of *number*. e is 2.7182818..., the base of the natural logarithm.

EXP is the inverse of the LN function (natural logarithm).

To calculate powers of other bases, use the exponentiation operator (^).

Examples

EXP(1) *equals* 2.7182818 (the value of e)

EXP(LN(3)) *equals* 3

FALSE

FALSE( )

The FALSE function gives the logical value FALSE.

Example

CHOOSE(4,FALSE( ),TRUE( ),TRUE( ),FALSE( ),TRUE( ))
equals FALSE

FIXED(number,number-of-digits)

FIXED

The FIXED function rounds *number* to *number-of-digits*, formats it in decimal format with commas, and gives the result as text.

If *number-of-digits* is negative, *number* is rounded to the left of the decimal point. If you omit *number-of-digits*, it is assumed to be 2.

Examples

FIXED(1234.567,2) *equals* "1,234.57"

FIXED(−1234.567,−2) *equals* "−1,200"

FIXED(0.123,4) *equals* "0.1230"

FV(rate,nper,pmt,pv,type)

FV

The FV function gives the future value of an investment involving constant cash flows.

If you omit either *pv* or *type*, it is assumed to be 0.

For a complete description of the arguments in FV, see PV.

GROWTH(Y-array,X-array,x-array)

GROWTH

The GROWTH function gives, as an array, the y values on the exponential curve of regression, y=b*m^x, for two variables, X and Y, represented by *X-array* and *Y-array*. The y values correspond to the x values in *x-array*, and the array of y values is of the same size and shape as *x-array*. The curve of regression is determined by the same method used for LOGEST.

The size and shape of *X-array* should be the same as those of *Y-array*. If you omit *x-array*, it is assumed to be the same as *X-array*. If you omit both *X-array* and *x-array*, both are assumed to contain 1,2,3,... . If any of the numbers in *Y-array* are negative, GROWTH gives the error value #NUM!.

GROWTH is the analog for exponential curves of the TREND function for lines.

HLOOKUP
VLOOKUP

HLOOKUP(lookup-value,compare-array,index-number)
VLOOKUP(lookup-value,compare-array,index-number)

The HLOOKUP function searches the first row of *compare-array* for the largest value which is less than or equal to *lookup-value*. Having found that value in some column of the first row, HLOOKUP moves down in that column by an amount specified by *index-number* and gives the value found there.

Index-number must be greater than or equal to 1. If *index-number* is 1, HLOOKUP gives the value from the first row. If *index-number* is negative, HLOOKUP gives the error value #VALUE!. If *index-number* is greater than the number of rows in *compare-array*, HLOOKUP gives the error value #REF!.

The values in the first row of *compare-array* can be text, numbers, or logical values. They must be in ascending order, according to Excel's sorting order of numbers < text < logical values. If *lookup-value* is smaller than the smallest value in the first row of *compare-array*, HLOOKUP gives the error value #N/A.

The VLOOKUP function is identical to HLOOKUP, except that it searches the first column of *compare-array*, and moves right in that row by an amount specified by *index-number*.

Examples

If A1:E1 contains the comparison values $10,000, $20,000, $30,000, $40,000, and $50,000, and A5:E5 contains the values 10%, 20%, 30%, 40%, and 50%, then:

HLOOKUP(35000,A1:E5,5) *equals* 30%

HLOOKUP(35000,A1:E5,1) *equals* 30000

HLOOKUP(5000,A1:E5,5) *equals* #N/A

HOUR
MINUTE
SECOND

HOUR(serial-number)
MINUTE(serial-number)
SECOND(serial-number)

The HOUR function gives the hour in *serial-number* as an integer in the range 0 to 23. The MINUTE and SECOND functions give the minute and second as integers in the range 0 to 59. For an explanation of times and serial numbers, see "Entering a Date or Time" in Chapter 4 of the *User's Guide*.

HOUR, MINUTE, and SECOND use only the fractional part of *serial-number*, since times are represented as decimal fractions in the range 0 to 1, not including 1. You can give *serial-number* as text, such as "16:48:00" or "4:48:00 PM", instead of as a number; Excel automatically converts the text into the serial number.

Examples

HOUR(0.7) *equals* 16

MINUTE("4:48:00 PM") *equals* 48

SECOND(TIME(16,48,0)) *equals* 0

IF(logical,value-if-true,value-if-false)

IF

The IF function gives *value-if-true* if *logical* is TRUE, and *value-if-false* if *logical* is FALSE. If you omit *value-if-false*, IF gives FALSE if *logical* is FALSE.

Examples

Given that Oral is 98 and Written is 90:

IF(OR(Oral<80,Written<75),"Fail","Pass") *equals* "Pass"

Given that Choices is the array {"A","C","D","C"} and Answers is the array {"B","C","D","A"}:

IF(Choices=Answers,"Right","Wrong")
equals {"Wrong","Right","Right","Wrong"}

INDEX(ref,row,column,area)

INDEX (Form 1)

This INDEX function gives the reference of a single cell within *ref*, selected by the indices *row*, *column*, and *area*.

If *ref* consists of multiple areas on the worksheet, the *area* index selects one of the areas, the *column* index selects the column within that area, and the *row* index selects the row.

If *ref* is a single range, you can omit the *area* index. The *column* index selects the column within the range, and the *row* index selects the row.

If *ref* is one row high, you need only one index to select the column. Similarly, if *ref* is one column wide, you need only one index to select the row.

Row, *column*, and *area* should be positive integers. If *row* or *column* is 0, INDEX does not use it as an index, but gives the reference of an entire column or row within *ref*, rather than the reference of a single cell.

If *area* is 0, or if an index is negative, INDEX gives the error value #VALUE!. If an index is greater than the size of *ref*, INDEX gives the error value #REF!.

Because INDEX of a reference gives a reference, rather than the value within the referenced cell, you can use INDEX within other functions that take references as arguments or within reference formulas. However, when you use INDEX within a function that takes values as arguments, the reference produced by INDEX will be converted automatically to the value inside the referenced cell. See the last example below.

Examples

INDEX((B2:D4,F6:I9),3,4,2) *equals* I8

INDEX((B2:D4,F6:I9),3,4,1) *equals* #REF!

INDEX(B2:D4,2,3,1) *equals* D3

INDEX(B2:D4,2,3) *equals* D3

INDEX(B2:D2,1,2,1) *equals* C2

INDEX(B2:D2,2) *equals* C2

INDEX(B2:B4,2,1,1) *equals* B3

INDEX(B2:B4,2) *equals* B3

INDEX(B2:D4,2,0) *equals* B3:D3

INDEX(B2:D4,0,2) *equals* C2:C4

INDEX((B2:D4,F6:I9),0,0,1) *equals* B2:D4

ROW(INDEX(B2:D4,2,3)) *equals* 3

A1:INDEX(B2:D4,2,3) *equals* A1:D3

If A1:A5 contains the numbers 10, 7, 9, 27, and 2, then:

INDEX(A1:A5,3)/3 *equals* 3

INDEX (Form 2)

INDEX(array,row,column)

This INDEX function gives the value of a single element within *array*, selected by the indices *row* and *column*.

The *column* index selects the column within the array, and the *row* index selects the row. If *array* is one row high, you need only one index to select the column. Similarly, if *array* is one column wide, you need only one index to select the row.

Row and *column* should be positive integers. If *row* or *column* is 0, INDEX does not use it as an index, but gives, as an array, the values of an entire column or row within *array*, rather than the value of a single element.

If an index number is negative, INDEX gives the error value #VALUE!. If an index number is greater than the size of *array*, INDEX gives the error value #REF!.

Examples

INDEX({1,2,3;4,5,6},1,2) *equals* 2
INDEX({1,2,3;4,5,6},3,2) *equals* #REF!
INDEX({1;2;3;4;5},3,1) *equals* 3
INDEX({1;2;3;4;5},3) *equals* 3
INDEX({1,2,3,4,5},1,3) *equals* 3
INDEX({1,2,3,4,5},3) *equals* 3
INDEX({1,2,3;4,5,6},2,0) *equals* {4,5,6}
INDEX({1,2,3;4,5,6},0,2) *equals* {2;5}

INT(number)

INT

The INT function gives the largest integer less than or equal to *number*.

Examples

INT(8.9) *equals* 8
INT(−8.9) *equals* −9

IRR(values,guess)

IRR

The IRR function gives the internal rate of return of a series of cash flows, represented by the numbers in *values*. The internal rate of return is the interest rate that gives the series of cash flows a net present value of zero.

Excel uses an iterative technique to compute IRR, which can have more than one solution. If the successive results of IRR do not converge to within 0.0000001 after 20 iterations, IRR gives the error value #NUM!.

Guess is an optional argument that specifies the starting value of the iteration. If you omit *guess*, it is assumed to be 0.1 or 10%. If IRR does not converge, try different values for *guess*. Usually, a starting value between 0 and 1 will yield a meaningful result.

Values should be an array or reference that contains numbers. If *values* contains text, logical, or blank values, those values are ignored.

Example

If you have an investment represented by the cash flows (\$5000), \$1000, \$1500, \$2000, (\$3000), \$2500, \$5000, which are entered in A1:A7, then:

IRR(A1:A7,10%) *equals* 15.05%

ISERROR

ISERROR(value)

The ISERROR function gives the logical value TRUE if *value* is any Excel error value (#N/A, #VALUE!, #REF!, #DIV/0!, #NUM!, #NAME?, or #NULL!). Otherwise, it gives the logical value FALSE.

Example

Given that A1 contains the formula =3/0:

IF(ISERROR(A1),"Invalid Divisor","") *equals* "Invalid Divisor"

ISNA

ISNA(value)

The ISNA function gives the logical value TRUE if *value* is the error value #N/A. Otherwise, it gives the logical value FALSE.

Example

Given that A1 contains the formula =NA():

IF(ISNA(A1),"Value Not Available","")
equals "Value Not Available"

ISREF(value)

ISREF

The ISREF function gives the logical value TRUE if *value* is a reference or reference formula. Otherwise, it gives the logical value FALSE.

Examples

ISREF(A1) *equals* TRUE

ISREF(A1+B2) *equals* FALSE

If the name Sales is defined to be A1:C3, then:

ISREF(Sales) *equals* ISREF(A1:C3) *equals* TRUE

LEN(text)

LEN

The LEN function gives the number of characters in *text.*

Examples

LEN("Phoenix, AZ") *equals* 11

LEN("") *equals* 0

LINEST(Y-array,X-array)

LINEST

The LINEST function gives, as a horizontal array of two elements, the slope and y-intercept of the line of regression, $y = mx + b$, for two random variables, X and Y, represented by *X-array* and *Y-array*. The size and shape of *X-array* should be the same as those of *Y-array*. If you omit *X-array*, it is assumed to contain the numbers 1,2,3,... .

The line of regression is determined by the method of least squares.

The formulas used are:

$$m = \frac{n*(\sum(x*y))-(\sum x)*(\sum y)}{n*(\sum(x^2))-(\sum x)^2}$$

$$b = \frac{(\sum y)*(\sum(x^2))-(\sum x)*(sum(x*y))}{n*(\sum(x^2))-(\sum x)^2}$$

$$= AVERAGE(Y\text{-}array) - m*AVERAGE(X\text{-}array)$$

Examples

Suppose your monthly sales figures for the first six months of the year were $3100, $4500, $4400, $5400, $7500, and $8100, and those amounts are entered in A1:F1. The trend of your sales has slope and y-intercept given by:

LINEST(A1:F1,{1,2,3,4,5,6})
equals LINEST(A1:F1) *equals* {1000,2000}

If you want to project your sales in the ninth month of the year:

SUM(LINEST(A1:F1)*{9,1}) *equals* 11000

LN

LN(number)

The LN function calculates the natural logarithm of *number*. Natural logarithms use the mathematical constant e, 2.7182818..., as a base.

Number must be positive.

LN is the inverse of the EXP function, e raised to the power *number*.

Examples

LN(2.7182818) *equals* 1

LN(EXP(3)) *equals* 3

LN(8)/LN(2) *equals* 3 (the base 2 logarithm of 8)

LOG10

LOG10(number)

The LOG10 function gives the base 10 logarithm of *number*.

Number must be positive.

LOG10 is the inverse of base 10 exponentiation or the letter E in scientific notation.

Examples

LOG10(10) *equals* 1

LOG10(1E5) *equals* 5

LOG10(10^5) *equals* 5

LOG10(8)/LOG10(2) *equals* 3 (the base 2 logarithm of 8)

LOGEST(Y-array,X-array)

LOGEST

The LOGEST function gives, as a horizontal array of two elements, the parameters m and b of the exponential curve of regression, $y = b * m^x$, for two variables, X and Y, represented by *X-array* and *Y-array*. If you omit *X-array*, it is assumed to contain 1,2,3,... .

The size and shape of *X-array* should be the same as those of *Y-array*. If any of the numbers in *Y-array* are negative, LOGEST gives the error value #NUM!.

The curve of regression is determined by taking the natural logarithm of *Y-array*, applying the LINEST function to obtain a slope and y-intercept, and then raising e to those powers with the EXP function to produce m and b.

LOGEST is the analog for exponential curves of the LINEST function for lines.

LOOKUP(lookup-value,compare-vector,result-vector)

LOOKUP (Form 1)

This LOOKUP function searches *compare-vector* for the largest value that is less than or equal to *lookup-value*. LOOKUP gives the corresponding value in *result-vector*.

The values in *compare-vector* can be text, numbers, or logical values. They must be in ascending order, according to Excel's sorting order of numbers < text < logical values. If *lookup-value* is smaller than the smallest value in *compare-vector*, LOOKUP gives the error value #N/A. *Compare-vector* and *result-vector* should have the same length, but they need not have the same direction.

This form of LOOKUP is preferable to the form described below, which sometimes searches the first row of *compare-array* and sometimes searches the first column, depending upon the size of *compare-array*. Form 2 is included below because of its use in Microsoft Multiplan®.

Examples

If A1:E1 contains the comparison values $10,000, $20,000, $30,000, $40,000, and $50,000, and A4:E4 contains the values 10%, 20%, 30%, 40%, and 50%, then:

LOOKUP(35000,A1:E1,A4:E4) *equals* 30%

LOOKUP(5000,A1:E1,A4:E4) *equals* #N/A

LOOKUP(35000,A1:E1,A1:E1) *equals* 30000

LOOKUP (Form 2)

LOOKUP(lookup-value,compare-array)

This LOOKUP function searches the first row or column of *compare-array* for the largest value that is less than or equal to *lookup-value*. LOOKUP returns the corresponding value in the last row or column of *compare-array*.

Whether LOOKUP searches the first row or the first column of *compare-array* is determined by the size of *compare-array*. If *compare-array* is square or has more rows than columns, LOOKUP searches the first column and gives the value of the corresponding cell in the last column. If *compare-array* has more columns than rows, LOOKUP searches the first row and gives the value of the corresponding cell in the last row.

The values in the first row or column of *compare-array* can be text, numbers, or logical values. They must be in ascending order, according to Excel's sorting order of numbers < text < logical values. If *lookup-value* is smaller than the smallest value in the first row or column of *compare-array*, LOOKUP gives the error value #N/A.

Example

If A1:E1 contains the comparison values $10,000, $20,000, $30,000, $40,000, and $50,000, and A4:E4 contains the values 10%, 20%, 30%, 40%, and 50%, then:

LOOKUP(35000,A1:E4) *equals* 30%

MATCH

MATCH(lookup-value,compare-vector,type)

The MATCH function gives the number of the comparison value in *compare-vector* that matches *lookup-value*. If *lookup-value* matches the first comparison value, MATCH gives 1. If *lookup-value* matches the second comparison value, MATCH gives 2, and so on. If *lookup-value* does not match any of the comparison values, MATCH gives the error value #N/A.

Lookup-value can be a number, text, or a logical value. *Compare-vector* should be an array containing numbers, text, or logical values.

The rules for matching the values are determined by *type*. If *type* is 1, *lookup-value* matches the largest comparison value that is less than or equal to *lookup-value*. If *lookup-value* is less than the smallest comparison value, then it does not match any of them. The comparison values should be in ascending order, according to Excel's sorting order of numbers < text < logical values. If you omit *type*, it is assumed to be 1.

If *type* is −1, *lookup-value* matches the smallest comparison value that is greater than or equal to *lookup-value*. If *lookup-value* is greater than the largest comparison value, then it does not match any of them. The comparison values should be in descending order, according to Excel's sorting order of numbers < text < logical values.

If *type* is 0, *lookup-value* matches the first comparison value that is equal to *lookup-value*. If *lookup-value* is not equal to any of the comparison values, then it does not match any of them. If *lookup-value* is text, it may contain the wildcard characters * and ?. The comparison values can be in any order.

Examples

If A1:A5 contains the numbers 60, 70, 80, 90, and 100, then:

MATCH(75,A1:A5,1) *equals* 2

MATCH(75,A1:A5) *equals* 2

MATCH(55,A1:A5) *equals* #N/A

If A1:A5 contains the numbers 100, 90, 80, 70, and 60, then:

MATCH(75,A1:A5,−1) *equals* 3

MATCH(105,A1:A5,−1) *equals* #N/A

If A1:A5 contains the text Johnson, Nixon, Ford, Carter, and Reagan, then:

MATCH("Nixon",A1:A5,0) *equals* 2

MATCH("Truman",A1:A5,0) *equals* #N/A

MAX

MAX(numbers-1,numbers-2,...)

The MAX function gives the largest number in the list of arguments.

MAX can have from 1 to 14 arguments. The arguments should be numbers, or arrays or references that contain numbers. If an array or reference argument contains text or logical values, those values are ignored. If the arguments contain no numbers, MAX gives 0.

Examples

If A1:A5 contains the numbers 10, 7, 9, 27, and 2, then:

MAX(A1:A5) *equals* 27

MAX(A1:A5,30) *equals* 30

MID

MID(text,start-position,number-of-characters)

The MID function extracts *number-of-characters* from *text,* starting with the character in *start-position.* (The first character in *text* is considered to be in position 1.)

If *number-of-characters* plus *start-position* is greater than the length of *text*, MID returns the last characters in *text* beginning with the character in *start-position.* If *start-position* is less than 1 or if *number-of-characters* is negative, MID gives the error value #VALUE!.

Examples

MID("Hello world",1,5) *equals* "Hello"

MID("Hello world",12,5) *equals* "" (empty text)

If Grade is 85, then:

MID("FFFFFFDCBAA",INT(Grade/10)+1,1) *equals* "B"

MIN(numbers-1,numbers-2,...)

MIN

The MIN function gives the smallest number in the list of arguments.

MIN can have from 1 to 14 arguments. The arguments should be numbers, or arrays or references that contain numbers. If an array or reference argument contains text or logical values, those values are ignored. If the arguments contain no numbers, MIN gives 0.

Examples

If A1:A5 contains the numbers 10, 7, 9, 27, and 2, then:

MIN(A1:A5) *equals* 2

MIN(A1:A5,0) *equals* 0

MINUTE(serial-number)

MINUTE

See HOUR.

MIRR(values,safe,risk)

MIRR

The MIRR function gives the modified internal rate of return of a series of cash flows, represented by the numbers in *values*, given *safe* and *risk*. *Safe* is the rate returned by the investment that will finance the negative cash flows. *Risk* is the rate at which the positive cash flows can be reinvested.

Values should be an array or reference that contains numbers. If *values* contains text, logical, or blank values, those values are ignored.

Where n is the number of cash flows in *values*, the formula used is:

$$\left(\frac{-NPV(risk,values[positive])*(1+risk)^{n}}{NPV(safe,values[negative])*(1+safe)}\right)^{\frac{1}{n-1}}-1$$

Example

If you have an investment represented by the cash flows (\$5000), \$1000, \$1500, \$2000, (\$3000), \$2500, and \$5000, entered in A1:A7, and you hope to finance your negative cash flows with money borrowed at 12% and reinvest your positive cash flows at 17%, then:

MIRR(A1:A7,12%,17%) *equals* 15.19%

MOD

MOD(number,divisor-number)

The MOD function gives the remainder (modulus) after *number* is divided by *divisor-number*. The result has the same sign as *divisor-number*.

If *divisor-number* is 0, MOD gives the error value #DIV/0!.

Examples

MOD(3,2) *equals* MOD(−3,2) *equals* 1

MOD(3,−2) *equals* MOD(−3,−2) *equals* −1

MOD(7,4) *equals* 7−INT(7/4)*4 *equals* 3

MONTH

MONTH(serial-number)

See DAY.

NA

NA( )

The NA function gives the error value #N/A.

Examples

Given that A1 contains the formula =NA( ):

IF(ISNA(A1),"Value Not Available","")
equals "Value Not Available"

NOT(logical)

NOT

The NOT function gives the logical value FALSE if *logical* is TRUE, and gives TRUE if *logical* is FALSE.

Examples

NOT(1 + 1 = 2) *equals* FALSE

NOT(1 + 1 = 3) *equals* TRUE

NOW( )

NOW

The NOW function gives the serial number of the current date and time, updated at every recalculation. For an explanation of dates, times, and serial numbers, see "Entering a Date or Time" in Chapter 4 of the *User's Guide*.

The integer part of the serial number is in the range 0 to 49710, representing dates from January 1, 1904 to February 6, 2040. The fractional part is in the range 0 to 0.999, representing times from 0:00:00 or 12:00:00 AM to 23:59:59 or 11:59:59 PM.

NPER(rate,pmt,pv,fv,type)

NPER

The NPER function gives the number of periods of an investment involving constant cash flows.

If you omit either *fv* or *type*, it is assumed to be 0.

For a complete description of the arguments in NPER, see PV.

NPV(rate,values-1,values-2,...)

NPV

The NPV function gives the net present value of a series of future cash flows, represented by the numbers in the list of values, discounted at a constant interest rate specified by *rate*.

The cash flows are assumed to occur at equal time intervals, the first cash flow occurring at the end of the first period.

The list of values can contain from 1 to 14 arguments. The *values* arguments should be numbers, or arrays or references that contain numbers. If an array or reference argument contains text or logical values, those values are ignored.

Where n is the number of cash flows in the *values* arguments, the formula used is:

$$\sum_{i=1}^{n} \frac{values_i}{(1+rate)^i}$$

To find the net present value of a series of cash flows whose first flow occurs at the beginning of the first period, you must add the first flow separately.

Example

Assume you make an initial investment of $80000. If A1:A5 contains your expected returns at the end of each of the next five years, $5000, $15000, $30000, $40000, and $50000, and if the annual interest rate is 10%, the net present value of your investment is calculated by:

NPV(10%,A1:A5)−80000 *equals* 17848.20

OR

OR(logicals-1,logicals-2,...)

The OR function gives the logical value TRUE if any of the logical values in the list of arguments is TRUE. If all of the logical values are FALSE, OR gives the logical value FALSE.

OR can have from 1 to 14 arguments. The arguments should be logical values, or arrays or references that contain logical values. If an array or reference argument contains text or number values, those values are ignored.

Examples

OR(1+1=2,2+2=5) *equals* TRUE

OR(1+1=3,2+2=5) *equals* FALSE

PI

PI( )

The PI function gives the number 3.14159... , an approximation of the mathematical constant π.

Example

ATAN(1)*180/PI( ) *equals* 45 (degrees)

PMT(rate,nper,pv,fv,type)

PMT

The PMT function gives the periodic payment on an investment involving constant cash flows.

If you omit either *fv* or *type*, it is assumed to be 0.

For a complete description of the arguments in PMT, see PV.

PV(rate,nper,pmt,fv,type)
FV(rate,nper,pmt,pv,type)
NPER(rate,pmt,pv,fv,type)
PMT(rate,nper,pv,fv,type)
RATE(nper,pmt,pv,fv,type,guess)

PV
FV
NPER
PMT
RATE

Present value (*pv*), future value (*fv*), number of periods (*nper*), periodic payment (*pmt*), and interest rate per period (*rate*) are the five standard financial parameters in cash flow problems involving constant payments. Each of the above functions gives the value of one of these financial arguments, given the values of the others.

For the arguments *pv*, *fv*, and *pmt*, use the following cash flow convention: cash received is represented by a positive number; cash paid out is represented by a negative number. The examples below illustrate this convention.

Rate and *nper* must refer to the same period. For example, if *nper* is the number of months, then *rate* must be the effective monthly interest rate.

Type indicates whether payments occur at the beginning or the end of the periods. If *type* is 0, the first payment is assumed to occur at the end of the first period; if *type* is 1, the first payment occurs at the beginning. If you omit *type*, it is assumed to be 0.

Excel uses an iterative technique to compute RATE, which can have zero, one, or two solutions. If the successive results of RATE do not converge to within 0.0000001 after 20 iterations, RATE gives the error value #NUM!.

Guess is an optional argument that specifies the starting value of the iteration. If you omit *guess*, it is assumed to be 0.1 or 10%. If RATE does not converge, try different values for *guess*. Usually, a starting value between 0 and 1 will yield a meaningful result.

All of the arguments must be numbers. In PV, FV, NPER, and PMT, you can omit the last two arguments; in RATE, you can omit the last three arguments. Any argument you omit is assumed to be 0.

Excel uses the following equations to solve for one financial argument in terms of the others:

$$pv*(1+rate)^{nper}+pmt*(1+rate*type)*\frac{(1+rate)^{nper}-1}{rate}+fv=0$$

$$(for\ \ rate<>0)$$

$$pv+pmt*nper+fv=0$$

$$(for\ \ rate=0)$$

Examples

You have received a loan of $12,000 at an interest rate of 9% (monthly rate of 0.75%), and you must pay it off in 24 monthly payments made at the end of each month. *Pv* is 12000, *rate* is 0.75%, *nper* is 24, and *fv* and *type* are 0. The monthly payment is calculated by:

PMT(0.75%,24,12000) *equals* −548.22

A limited partnership plans to buy an apartment building for $750,000 with the expectation of selling the building for $1,500,000 in 6 years. The rent from the apartments amounts to $100,000 annually. *Pv* is −750000, *fv* is 1500000, *nper* is 6, *pmt* is 100000, and *type* is 0. If your guess at the annual rate of return is 10%, the actual annual rate of return is calculated by:

RATE(6,100000,−750000,1500000,0,10%) *equals* 22.73%

You now have $1,000 in your savings account earning 6% annual interest (0.5% monthly interest), and you plan to deposit $100 at the beginning of every month for the next 12 months. *Pv* is −1000, *rate* is 0.5%, *pmt* is −100, *nper* is 12, and *type* is 1, so the amount of money in your savings account at the end of 12 months is calculated by:

FV(.5%,12,−100,−1000,1) *equals* 2301.40

RAND( )

RAND

The RAND function gives a random number in the range 0 to 0.999... . RAND generates a new random number every time the sheet is recalculated.

RATE(nper,pmt,pv,fv,type,guess)

RATE

The RATE function gives the interest rate per period of an investment involving constant cash flows.

If you omit any one of *fv*, *type*, or *guess*, it is assumed to be 0.

For a complete description of the arguments in RATE, see PV.

REPT(text,number-of-times)

REPT

The REPT function takes *text* and repeats it *number-of-times* to make a new text value.

The maximum length of the resultant text is 255 characters. *Number-of-times* must be greater than or equal to 0. If it is 0, REPT gives the empty text.

Example

REPT("*-",3) *equals* "*-*-*-"

ROUND(number,number-of-digits)

ROUND

The ROUND function gives *number* rounded to *number-of-digits*. If *number-of-digits* is positive, *number* is rounded to that many decimal places. If *number-of-digits* is 0, *number* is rounded to the nearest integer. If *number-of-digits* is negative, *number* is rounded to the left of the decimal point.

Examples

ROUND(2.15,1) *equals* 2.2

ROUND(2.149,1) *equals* 2.1

ROUND(−1.475,2) *equals* −1.48

ROUND(891,−2) *equals* 900

ROW

ROW(ref)

The ROW function gives the row number of *ref*, if *ref* is the reference of a single cell. If *ref* is the reference of a range of cells, ROW gives the row numbers of *ref* as a vertical array. If *ref* is omitted, it is assumed to be the reference of the cell(s) in which the ROW function appears.

Ref cannot be a reference to multiple areas.

Examples

ROW(A3) *equals* 3

ROW(A3:B5) *equals* {3;4;5}

If ROW is entered in C5, then:

ROW( ) *equals* ROW(C5) *equals* 5

ROWS

ROWS(array)

The ROWS function gives the number of rows in *array*.

Examples

ROWS(A1:C4) *equals* 4

ROWS({1,2,3;4,5,6}) *equals* 2

SEARCH

SEARCH(search-text,text,start-number)

The SEARCH function searches for *search-text* within *text*, starting the search at the character specified by *start-number*. The first character in *text* is considered to be character number 1. If *search-text* appears within *text*, SEARCH gives the number of the character at which *search-text* first occurs. If *search-text* does not appear within *text*, SEARCH gives the error value #VALUE!.

If *start-number* is less than or equal to zero or greater than the number of characters in *text*, SEARCH gives the error value #VALUE!. If you omit *start-number*, it is assumed to be 1, and the search starts at the first character of *text*. If *search-text* is the empty text, it will match the first character that is searched, the character numbered *start-number*.

While searching for *search-text* within *text*, SEARCH ignores case distinction between *search-text* and *text*. *Search-text* may contain the wildcard characters ? and *. The question mark in a *text* indicates that any character can occupy that position. The asterisk indicates that any sequence of characters of any length can occupy that position.

Examples

SEARCH(" ","Hello world",1)
equals SEARCH(" ","Hello world") *equals* 1

SEARCH("cys-???-cys","val-leu-cys-his-cys-phe") = 9

SEARCH("i","Mississippi",SEARCH("i","Mississippi")+1) = 5

SECOND(serial-number)

SECOND

See HOUR.

SIGN(number)

SIGN

The SIGN function gives 1 if *number* is positive, 0 if *number* is 0, and −1 if *number* is negative.

Examples

SIGN(7+3) *equals* 1

SIGN(4−4) *equals* 0

SIGN(3−7) *equals* −1

SIN(number)

SIN

The SIN function gives the sine of *number*, where *number* is an angle in radians.

Examples

SIN(1.047) *equals* 0.866

SIN(60*PI()/180) *equals* 0.866

SQRT

SQRT(number)

The SQRT function gives the positive square root of *number*.

Number must be greater than or equal to 0. If *number* is negative, SQRT gives the error value #NUM!.

STDEV

STDEV(numbers-1,numbers-2,...)

The STDEV function gives the sample standard deviation of the numbers in the list of arguments.

STDEV can have from 1 to 14 arguments. The arguments should be numbers, or arrays or references that contain numbers. If an array or reference argument contains text, logical, or blank values, those values are ignored.

The formula used is:

SQRT(*VAR*(*numbers*-1,*numbers*-2,...))

The sample standard deviation is the best estimate of the population standard deviation based on a sample of the population. If your sample represents the entire population, you can compute the true population standard deviation by including the average of the sample in the list of arguments to STDEV:

STDEV(*Population*,*AVERAGE*(*Population*))

Examples

If A1:A5 contains the numbers 10, 7, 9, 27, and 2, then:

STDEV(A1:A5) *equals* 9.460

STDEV(A1:A5,AVERAGE(A1:A5)) *equals* 8.462

SUM

SUM(numbers-1,numbers-2,...)

The SUM function gives the sum of the numbers in the list of arguments.

SUM can have from 1 to 14 arguments. The arguments should be numbers, or arrays or references that contain numbers. If an array or reference argument contains text, logical, or blank values, those values are ignored.

Examples

If A1:A5 contains the numbers 10, 7, 9, 27, and 2, then:

SUM(A1:A5) *equals* 55

SUM(A1:A5,5) *equals* 60

TAN(number)

TAN

The TAN function gives the tangent of *number*, where *number* is an angle in radians.

Examples

TAN(0.785) *equals* 1

TAN(45*PI()/180) *equals* 1

TEXT(number,format-text)

TEXT

The TEXT function formats *number* as specified by *format-text*, and gives the result as text.

Number is formatted according to the rules used by the Number command from the Format menu. *Format-text* cannot contain an asterisk (*) and cannot be the General format.

Examples

TEXT(2.715,"$0.00") *equals* "$2.72"

TEXT("4/15/1985","mmmm d, yyyy") *equals* "April 15, 1985"

TIME(hour,minute,second)

TIME

The TIME function gives the serial number of the time specified by *hour*, *minute*, and *second*. For an explanation of times and serial numbers, see "Entering a Date or Time" in Chapter 4 of the *User's Guide*.

The serial number is a decimal fraction in the range 0 to 1, not including 1, representing times from 0:00:00 or 12:00:00 AM to 23:59:59 or 11:59:59 PM. *Hour*, *minute*, and *second* should make up a valid time in this range.

It is not necessary to enter a formula like =TIME(16,48,0) in a cell; it is easier to enter the time constant 16:48:00 or 4:48:00 PM in the cell to produce the same serial number. Similarly, it is not necessary to use a formula like =TIME(16,48,0) within a larger formula, because it is easier to use the text "16:48:00" or "4:48:00 PM" in the formula; Excel automatically converts the text into the serial number.

However, if you want *hour*, *minute*, or *second* to be formulas rather than constants, you must use the TIME function.

Examples

TIME(12,0,0) *equals* 0.5

TIME(16,48,0)−TIME(12,0,0) *equals* 0.2

"16:48:00"−"12:00:00" *equals* 0.2

TRANSPOSE

TRANSPOSE(array)

The TRANSPOSE function gives, as an array, the transpose of *array*.

Examples

TRANSPOSE({1,2,3;4,5,6}) *equals* {1,4;2,5;3,6}

If A1:A5 contains the numbers 10, 7, 9, 27, and 2; that is, the array {10;7;9;27;2}, then:

TRANSPOSE(A1:A5) *equals* {10,7,9,27,2}

TREND

TREND(Y-array,X-array,x-array)

The TREND function gives, as an array, the y values on the line of regression, y = mx + b, for two random variables, X and Y, represented by *X-array* and *Y-array*. The y values correspond to the x values in *x-array*, and the array of y values is of the same size and shape as *x-array*. The line of regression is determined by the same method used for LINEST.

The size and shape of *X-array* should be the same as those of *Y-array*. If you omit *x-array*, it is assumed to be the same as *X-array*. If you omit both *X-array* and *x-array*, both are assumed to contain 1,2,3,... .

Examples

If your monthly sales figures for the first six months of the year are $3100, $4500, $4400, $5400, $7500, and $8100, entered in A1:F1, the trend of your sales is calculated by:

TREND(A1:F1,{1,2,3,4,5,6})
equals {3000,4000,5000,6000,7000,8000}

If you want to project your sales ahead to the next three months, then:

TREND(A1:F1,{1,2,3,4,5,6},{7,8,9}) *equals* {9000,10000,11000}

TRUE( )

TRUE

The TRUE function gives the logical value TRUE.

Example

CHOOSE(2,FALSE( ),TRUE( ),TRUE( ),FALSE( ),TRUE( ))
equals TRUE

TYPE(value)

TYPE

The TYPE function gives the type of *value*.

If *value* is	TYPE gives
A number	1
Text	2
A logical value	4
An error value	16
An array	64

Examples

TYPE({1,2;3,4}) *equals* 64

If A1 contains the text "Smith", then:

TYPE(A1) *equals* TYPE ("Smith") *equals* 2

TYPE("Mr."&A1) *equals* 2

TYPE(2+A1) *equals* TYPE(#VALUE!) *equals* 16

VALUE

VALUE(text)

The VALUE function converts *text*, a number in the form of text, into a number.

Text can be in any of the constant number, date, or time formats recognized by Excel in the formula bar. If it is not, VALUE gives the #VALUE! error value.

It is never necessary to use the VALUE function in a formula, since Excel automatically converts the text to numbers, as necessary.

Examples

VALUE("$1,000") *equals* 1000

VALUE("16:48:00") − VALUE("12:00:00")
equals "16:48:00" − "12:00:00" *equals* 0.2

VAR

VAR(numbers-1,numbers-2,...)

The VAR function gives the sample variance of the numbers in the list of arguments.

VAR can have from 1 to 14 arguments. The arguments should be numbers, or arrays or references that contain numbers. If an array or reference argument contains text, logical, or blank values, those values are ignored.

The formula used is:

$$\frac{n*(\sum(x^2))-(\sum x)^2}{n*(n-1)}$$

The sample variance is the best estimate of the population variance based on a sample of the population. If your sample represents the entire population, you can compute the true population variance by including the average of the sample in the list of arguments to VAR:

VAR(*Population*,*AVERAGE*(*Population*))

Examples

If A1:A5 contains the numbers 10, 7, 9, 27, and 2, then:

VAR(A1:A5) *equals* 89.5

VAR(A1:A5,AVERAGE(A1:A5)) *equals* 71.6

VLOOKUP(lookup-value,compare-array,index-number)

VLOOKUP

See HLOOKUP.

WEEKDAY(serial-number)

WEEKDAY

See DAY.

YEAR(serial-number)

YEAR

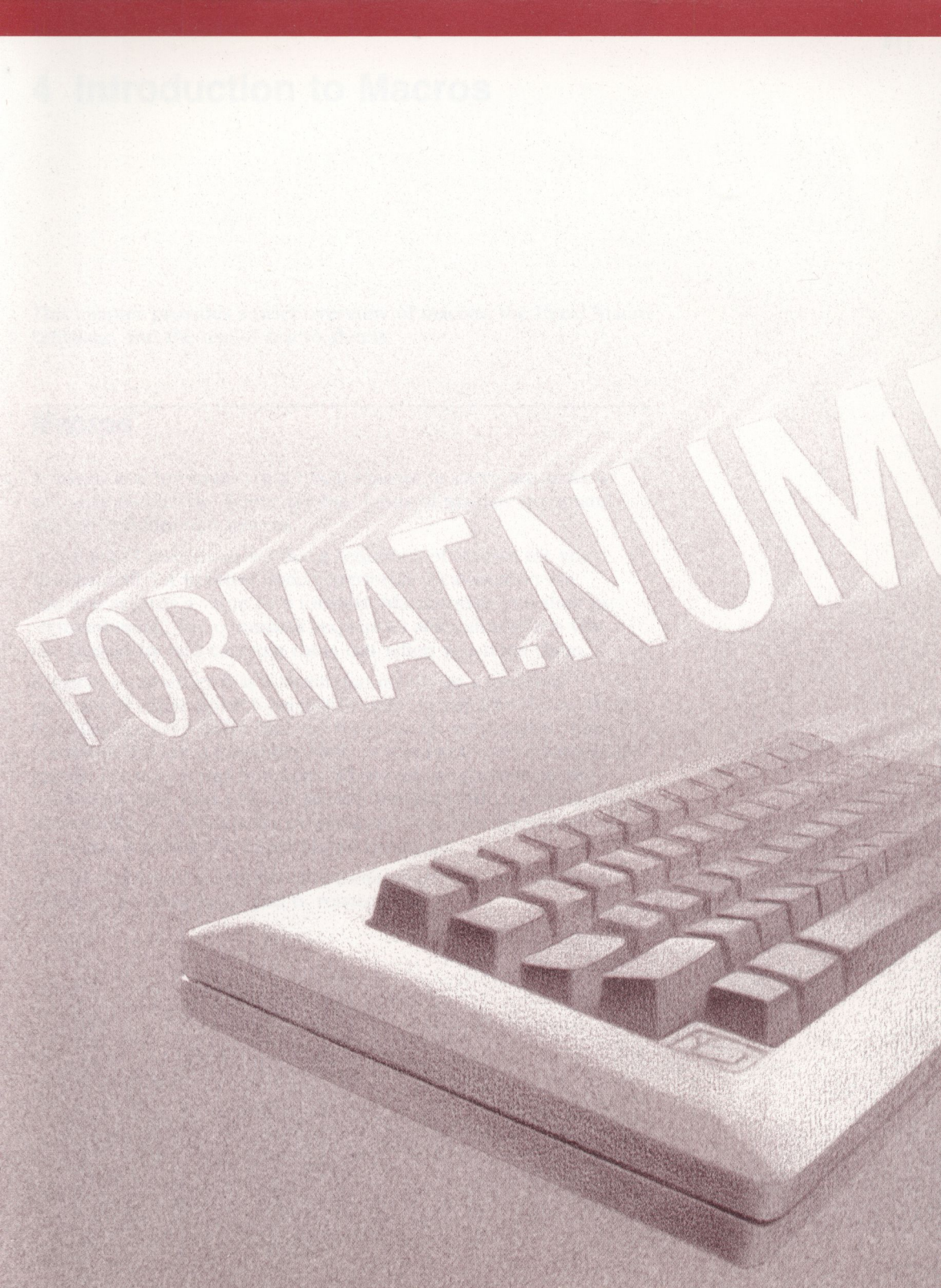
See DAY.

Macros

Macros

This part of the manual explains how to work with macros—one of the most useful aspects of Excel. After brief overviews of macros, the Excel Macro Language, and macro sheets, you will find three chapters that provide in-depth information on command macros, function macros, and macros sheets, respectively.

In addition, this part contains a concise listing of the Macro Language functions divided by menu and subject categories, followed by a detailed alphabetical listing of these functions.


FORMAT.NUM

4 Introduction to Macros

This chapter provides a brief overview of macros, the Excel Macro Language, and the use of macro sheets.

Macros

A macro is a "program" that Excel follows to carry out tasks or calculations for you. There are two kinds of macros: command macros and function macros.

A command macro carries out a sequence of actions for you automatically. A function macro performs a calculation for you and returns a value to a formula. Macros save you time by "remembering" the command sequences and calculations you use frequently and carrying them out when you need them.

A macro — whether it is a command macro or a function macro — is made up of formulas in cells on a macro sheet. The formulas are very much like the formulas on worksheets. They consist of the same elements, such as numbers, text, references, names, operators, and functions. However, in addition to the worksheet functions you are already familiar with, a special set of functions is available for creating macros. This special set of functions is called the Excel Macro Language.

For a complete description of command macros and function macros, see Chapters 5 and 6, respectively.

The Excel Macro Language

The Excel Macro Language is a special set of functions you can use to create macros.

Unlike worksheet functions, most Macro Language functions are able to take actions. That is, they can cause something to change on a document or in a window. The actions they take are the same kinds of actions you typically take with Excel, such as changing the selection on a worksheet, entering a formula in a cell, or formatting a number with the Number command from the Format menu. SELECT, FORMULA, and FORMAT.NUMBER are examples of Excel Macro Language functions that take an action.

Some action-taking functions (such as FORMAT.NUMBER) are the direct counterparts of commands in the Excel menu bar. These functions are sometimes referred to as command-equivalent functions. Other functions (such as SELECT) carry out actions you take without using a specific command. These are sometimes referred to as action-equivalent functions. Strictly speaking, action-taking functions also return a value. That value is TRUE if the action was successful and an error value if it was unsuccessful. However, when you use these functions, the returned value is of little concern.

There are a few Macro Language functions that do not take an action. Instead, they return a value to the macro sheet, just as a worksheet function returns a value to a worksheet. ABSREF, SELECTION, and GET.FORMULA are examples of Excel Macro Language functions that return a value to the macro sheet.

For a complete listing of all these functions, see Chapter 8, "Macro Language Function Directory."

Macro Sheets

In addition to worksheets and charts, Excel has a third kind of document called a macro sheet. A macro sheet looks like a worksheet, but is used for writing and storing macros.

In working with macro sheets, you use the same procedures that you use with worksheets. All the familiar tasks — such as entering formulas, editing cells, formatting cells, and saving documents — are the same for macro sheets as they are for worksheets.

There are, however, several differences between using worksheets and using macro sheets. One main difference while you're using macro sheets is that you have the functions from the Excel Macro Language at your disposal (see Chapter 8, "Macro Language Function Directory"). Another important difference is that Excel calculates the sheets differently (see "Macro Sheet Calculation" in Chapter 7, "More Information about Macro Sheets").

Other changes that occur when a macro sheet is active are the following:

- The Define Name command has some extra options.
- The list of functions in the Paste Function command contains the Macro Language functions.
- The Set Recorder command from the Macro menu is available for use.

To create a macro sheet

Creating a new macro sheet is just like creating a new worksheet.

1. Choose New from the File menu.
2. Click Macro Sheet.
3. Click OK.

For a detailed description of macro sheets and how they're used in Excel, see Chapter 7, "More Information about Macro Sheets."

5 Command Macros

This chapter describes how to use command macros in Excel. You'll learn how to record, name, and run a command macro, and see a detailed example of a typical one. Then, you'll learn how to edit an existing macro and how to make macros smart.

In addition, you'll find information about interactive command macros; and you'll find a brief introduction to how Excel calculates macro sheets, including information on the INPUT and GOTO functions.

What Is a Command Macro?

A command macro is a sequence of formulas that when calculated, carries out actions. The formulas are calculated when you run the macro.

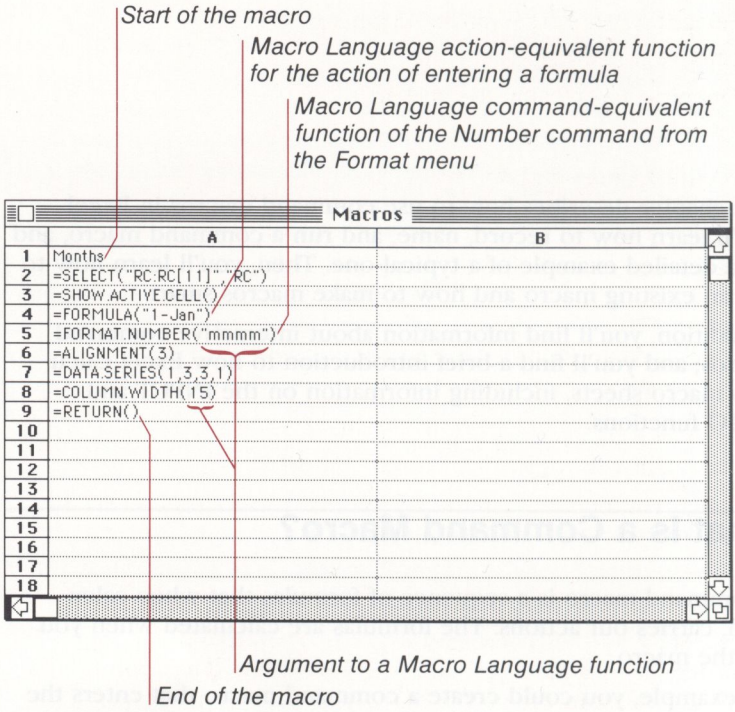
For example, you could create a command macro that enters the months, January through December, as column titles on your worksheet. You could name the macro Months and assign it the Option-Command-key m. Then, any time you want these months as column titles on a worksheet, you press Option-Command-m, and the macro enters them for you.

You can run a command macro from any worksheet by choosing the Run command from the Macro menu or by pressing an Option-Command-key that you assign to the macro.

Much more elaborate command macros are also possible. For example, your weekly routine may involve copying values from different worksheets onto a new worksheet and then printing the new worksheet. You could create a command macro called Summary that carries out these actions every time you select Summary with the Run command.

It is also possible to create a command macro that is interactive; it will pause in its execution and prompt you to supply information. After you enter the information, it continues executing.

Here's an example of what a command macro looks like:



There are three steps to working with command macros: entering, naming, and running the macro.

Entering a Command Macro

To enter a command macro, you enter the action-taking formulas in cells on a macro sheet. Excel provides a very easy method for entering these formulas. You can "record" a command macro by starting Excel's recorder and then carrying out the actions that you want the macro to perform. Excel automatically enters the action-taking functions from the Excel Macro Language to record your actions. To "play back" the macro, run it, and Excel calculates the formulas to carry out the actions.

Most of the actions you typically perform with Excel can be recorded, including:

- Selecting cells and entering formulas
- Choosing commands
- Opening and closing documents

To record a command macro:

To record a command macro

1. Create a new macro sheet or open an existing one.
2. Select the recording area (generally, a columnar range) you want for the macro and choose Set Recorder from the Macro menu.
3. Activate the worksheet on which you will carry out the actions to be recorded.
4. Choose Start Recorder from the Macro menu.
5. Carry out the actions you want to record.
6. When you finish the actions you want to record, choose Stop Recorder from the Macro menu.

Note Be prepared to begin recording immediately after you choose Start Recorder. From that moment until you choose Stop Recorder, every action you take will be recorded as a formula on the macro sheet in the recorder range. Think about exactly what you want to do before you start the recorder, so you can record the macro as efficiently as possible.

The actions you took while the recorder was on have been entered as formulas into the range of cells you designated with Set Recorder on the macro sheet. For a detailed illustration of this process, see "A Sample Command Macro" in this chapter.

Naming a Command Macro

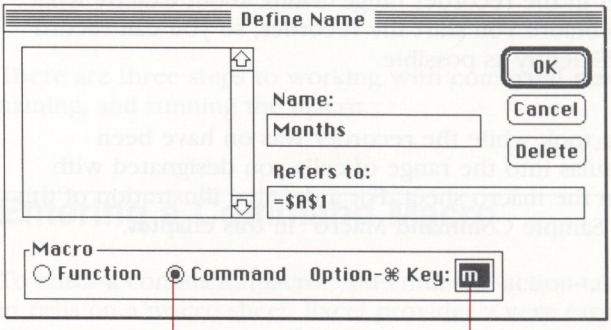
Before you use a command macro that you have recorded, you should define a name for it.

To name a command macro

To name a command macro:

1. Select the first cell of the command macro on the macro sheet.
2. Choose Define Name from the Formula menu. When a macro sheet is active, Define Name has some extra options, as shown below.
3. Type a name for the macro in the Name box.
4. Click Command.
5. If you want to be able to run the macro by pressing an Option-Command-key, type the key code you want in the box provided. For example, if you type m in the box, you can run the macro from a worksheet by pressing Option-Command-m. Note that Excel distinguishes between uppercase and lowercase Option-Command-keys. The Option-Command key code is optional.
6. Click OK.

The Define Name dialog box looks like this:



The Command button indicates that the named cell is at the start of a command macro.

The optional key code can be used later to run the command macro.

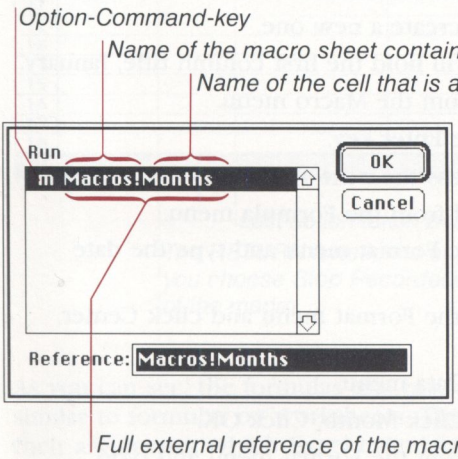
Running a Command Macro

There are two ways to run a command macro.

To run a command macro

1. Activate the worksheet from which you want to run the command macro.
2. Choose Run from the Macro menu. The dialog box displays a list box of all names you have defined and designated as "Command" names on all open macro sheets.
3. Select the name of the macro you want to run.
4. Click OK.

The Run dialog box looks like this:



Or,

1. Activate the worksheet from which you want to run the command macro.
2. Type the macro's Option-Command-key, if you assigned one with the Define Name command.

When you run a macro, Excel goes to the first cell of the macro and begins calculating the formulas. Calculation proceeds down the column(s) until a RETURN function ends the macro.

A Sample Command Macro

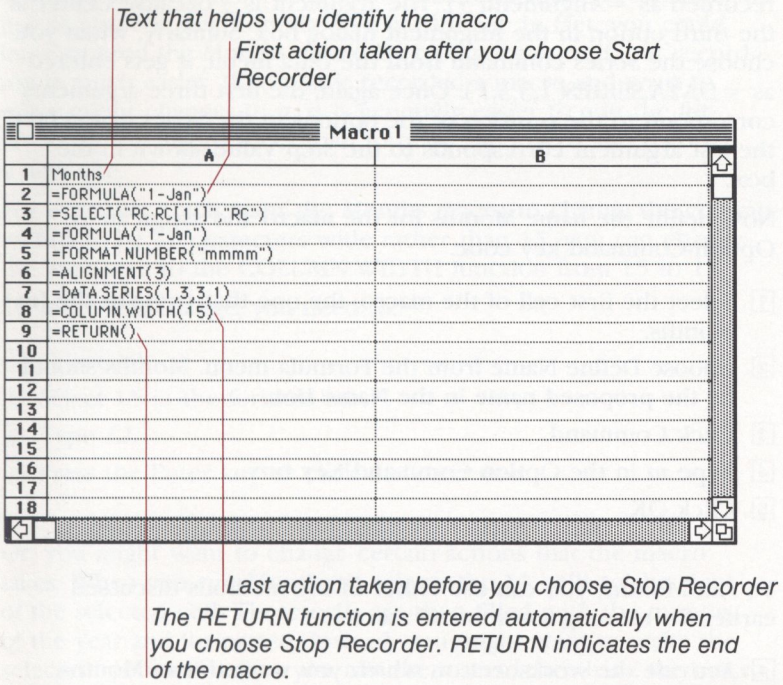
Here is a typical situation that calls for a command macro. Suppose you frequently enter the months, January through December, as column titles on your worksheets; and you always center the column titles and widen the columns to 15 characters. All of these actions can be carried out using a single command macro, as shown below. We will call the macro Months.

To record the Months macro

To record the Months macro:

1. Create a new macro sheet.
2. Select column A.
3. Choose Set Recorder from the Macro menu.
4. Type *Months* in A1 and press the Enter key. This is just a label to help you identify the macro later.
5. Activate a worksheet or create a new one.
6. Select A1, the cell that will hold the first column title, January.
7. Choose Start Recorder from the Macro menu.
8. Type *1-Jan* and press the Enter key.
9. Extend the selection across the next 11 cells in the row.
10. Choose Show Active Cell from the Formula menu.
11. Choose Number from the Format menu and type the date format *mmmm*. Click OK.
12. Choose Alignment from the Format menu and click Center. Click OK.
13. Choose Series from the Data menu.
14. Click Rows. Click Date. Click Month. Click OK.
15. Choose Column Width from the Format menu and type a column width of *15*. Click OK.
16. Choose Stop Recorder from the Macro menu.

The actions you took on the worksheet have been recorded as formulas on the macro sheet. Activate the macro sheet. The macro looks like this:



As you can see, the formulas that make up the macro are very similar to formulas on worksheets. They consist of functions and their arguments, which Excel calculates when carrying out the macro. The only difference is that the macro sheet formulas can use functions from the Excel Macro Language. In a recorded command macro, most of these functions are action-taking functions that cause something to happen or change on the calling document.

Every action you took during recording has a corresponding formula in a cell on the macro sheet. The arguments to each function describe what particular action the function takes. For example, the SELECT function takes the action of selecting cells, and its arguments specify which cells should be selected and which cell should be the active cell. In the COLUMN.WIDTH function, the argument specifies what width to use.

Specifically, when you enter 1-Jan, it gets recorded as =FORMULA("1-Jan") on the macro sheet. Here, the function is FORMULA, and the argument is 1-Jan. When you choose the Alignment command from the Format menu and click Center, it gets recorded as =Alignment(3). The argument is 3 because Center is the third option in the Alignment dialog box. Similarly, when you choose the Series command from the Data menu, it gets entered as =DATA.SERIES(1,3,3,1). Once again, the first three arguments correspond to the position of the options in the Series dialog box; the last argument corresponds to the Step Value shown in the box.

To name the Months macro

Now define the name Months for the macro and assign an Option-Command key code.

1. Select the first cell of the macro, the one that contains the text Months.
2. Choose Define Name from the Formula menu. Months should be the proposed name in the Name box.
3. Click Command.
4. Type *m* in the Option-Command Key box.
5. Click OK.

To run the Months macro

To run Months, you can use either of the methods discussed earlier for running a command macro.

1. Activate the worksheet on which you want to run Months.
2. Select the cell you want for the first column title, January.
3. Choose Run from the Macro menu. The dialog box displays a list box with all of the command macros that you have named on all open macro sheets and their Option-Command-keys.
4. Select Macro1!Months and click OK.

Or,

1. Activate the worksheet on which you want to run Months.
2. Select the cell you want for the first column title, January.
3. Press Option-Command-m to run the macro.

Editing a Macro

After you have created a command macro, you may want to change it slightly. It is possible to edit the formulas in a macro just as you edit any formula on a worksheet. In fact, you could have entered the Months macro by typing it yourself, but recording is much easier. If you have recorded a macro and want to make minor changes to it, it is generally easier to edit the formulas on the macro sheet than to erase the macro and record a new one.

For example, if you want the Months macro to change the column widths to be 12 characters wide rather than 15, you can change the argument to the COLUMN.WIDTH function from 15 to 12.

To make this change, you need to:

To change the Months macro

1. Select A8.
2. Select 15 in the formula bar.
3. Type *12*.
4. Press the Enter key or click the enter box.

Or, you might want to change certain actions that the macro takes. When you run Months, it selects the 11 cells to the right of the selected cell. These cells are then filled with the months of the year and formatted. Instead, you might want to make the selection yourself before you run the macro, then have the macro fill up only the selected cells with months and format them.

You can make this change by deleting the first two formulas from the macro, as shown below:

1. Select rows 3 and 4.
2. Choose Delete from the Edit menu.

The resulting macro looks like this:

Macro1

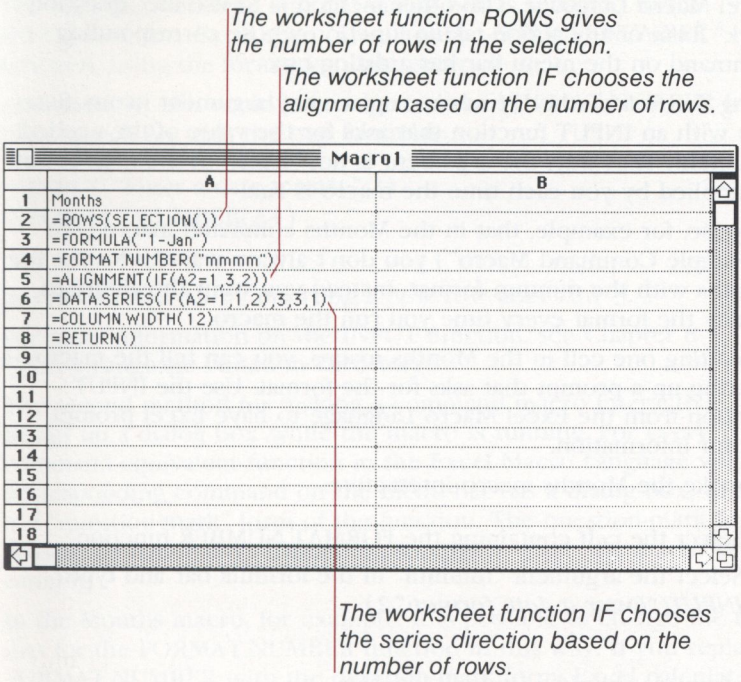
	A	B
1	Months	
2	=FORMULA("1-Jan")	
3	=FORMAT.NUMBER("mmmm")	
4	=ALIGNMENT(3)	
5	=DATA.SERIES(1,3,3,1)	
6	=COLUMN.WIDTH(12)	
7	=RETURN()	
8		
9		
10		
11		
12		
13		
14		
15		
16		
17		
18		

Smart Macros

So far, all of the macro formulas you have seen have contained Macro Language functions that take an action. There are also Macro Language functions that give information about Excel and the open documents. For example, the SELECTION function gives the reference of the selection. In addition, you can enter any worksheet function or formula on a macro sheet. By employing these functions, you can make your macros "smart."

For example, the edited Months macro works fine if the selection is a row of cells. But what if you sometimes want a row of months and sometimes want a column? If you have a column of months, you want the alignment to be left rather than centered. Also, the Series command must put the series in columns rather than rows. With the Macro Language function SELECTION, and the built-in worksheet functions IF, ROWS, and COLUMNS, the macro can determine whether the selection is a row or a column and then make a decision about the ALIGNMENT and DATA.SERIES functions accordingly.

A smart version of the Months macro looks like this:



In cells A5 and A6 above, the IF functions check to see if there is more than one row in the selection. In cell A5, if there is one row, IF chooses 3, the third option (Center) in the Alignment dialog box. Otherwise, it chooses 2, corresponding to the second option (Left). Similarly, in cell A6, if there is one row in the selection, IF chooses 1, the first option (Rows) in the Series dialog box. Otherwise, it chooses 2, corresponding to the second option (Columns).

Making Command Macros Interactive

An interactive command macro is one that pauses during execution and prompts you for information. The macro may display the dialog box of an Excel menu command, a Macro Language function dialog box that prompts you and waits for you to enter information, or an alert box. The information you enter is returned to the macro, and the macro continues executing.

There are two basic methods for making a command macro interactive. One method is to use the INPUT function from the Excel Macro Language. The other method is to use the "question-mark" form of any action-taking function whose corresponding command on the menu bar has a dialog box.

Using the INPUT function

Using the first method, you can replace any argument in any function with an INPUT function that asks for the value of the needed argument. That way, the argument is not "fixed" in the macro, but is supplied by you each time the macro is run.

Suppose, for example, that in the Months command macro (see "A Sample Command Macro") you don't always want to format the months with the mmmm format. Instead you want the ability to specify the format every time you run the macro.

By editing one cell in the Months macro, you can tell the macro to bring up a prompt that asks for the format. Use the INPUT function from the Excel Macro Language to have Excel prompt you for information.

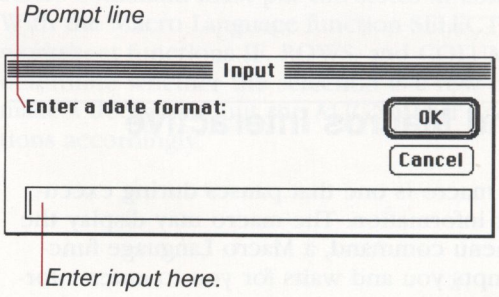
To make the Months macro interactive:

1. Select the cell containing the FORMAT.NUMBER function.
2. Select the argument "mmmm" in the formula bar and type *INPUT("Enter a date format:",2)*.

The formula should now look like this:

=FORMAT.NUMBER(INPUT("Enter a date format:",2))

When you run the command macro, Excel calculates the INPUT function. It displays the following dialog box and waits for you to enter information into the input box.



When the dialog box is displayed, type the desired date format and press the Enter key or click the OK button in the dialog box. The text that you type into the dialog box is returned to the INPUT function. Excel then calculates the FORMAT.NUMBER function, using the format you just entered.

Alternatively, you could have replaced the FORMAT.NUMBER formula in cell A4 with the two formulas:

```
=INPUT("Enter a date format:",2)
=FORMAT.NUMBER(A4)
```

in cells A4 and A5 by inserting a new row after row 3.

For more information on the INPUT function, see Chapter 8, "Macro Language Function Directory."

Using the question-mark form of a function

The second method for making a command macro interactive brings up a dialog box while the macro is running. For every command-equivalent function in the Excel Macro Language whose corresponding command on the menu bar has a dialog box, there is a "question-mark" form of the function. The question-mark form causes the command's dialog box to appear while the macro is running.

In the Months macro, for example, it is possible to change the format for the FORMAT.NUMBER function in this way. If you replace FORMAT.NUMBER with the question-mark form, Excel calculates the FORMAT.NUMBER? function, brings up the dialog box for the Number command, and pauses while you enter the format in the dialog box. When you press the Enter key or click the OK button, Excel carries out the command as specified and continues calculating the macro.

To see which functions have the question-mark form, see Chapter 8, "Macro Language Function Directory."

When you create an interactive macro, you usually want to prompt the person running the macro for the appropriate information. The INPUT function displays a prompt in its dialog box as specified by its first argument. However, the question-mark form of a command-equivalent function displays only the dialog box and not a prompt.

You can also make command macros interactive by using the ALERT function to bring up an alert box. For information on this function, see Chapter 8, "Macro Language Function Directory."

It is possible to have interactive command macros that are much more elaborate than Months. You can create "custom" applications that generate entire worksheets automatically while the user enters only the requested information. Such an application requires some investment in programming time, but it is a valuable tool for you and your co-workers, clients, or employees.

Calculating Macros

An important difference between a worksheet and a macro sheet is the way that Excel calculates each sheet. On a worksheet, Excel calculates all of the formulas that refer to a particular cell every time you edit the formula or value in that cell. On a macro sheet, Excel calculates the formulas in a macro according to a starting point, an ending point, and a set order of calculation. The formulas in a macro are not calculated until the macro is run.

Calculation begins in the first cell of a macro and proceeds down the column until a RETURN function or a HALT function ends the macro. This order of calculation is altered only when Excel encounters a GOTO function or a call to another macro which causes calculation to branch to another cell. Calculation then continues from that cell down its column.

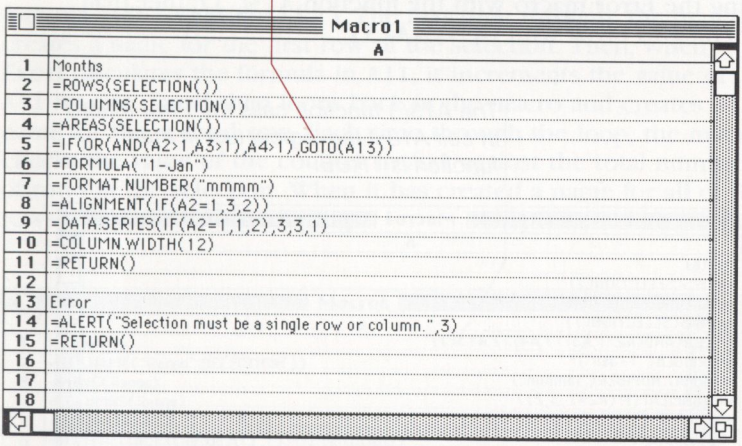
Branching with the GOTO function

Branching with the GOTO Function

The form of the GOTO function is GOTO(*ref*), where *ref* is a reference to another cell on a macro sheet. When Excel encounters a GOTO function, it continues calculation at, or "branches" to, the referenced cell. Calculation then proceeds down the column from that cell until a RETURN function ends the macro.

The GOTO function is particularly useful in conjunction with Excel's logical functions. For example, in the Months macro (see "A Sample Command Macro"), you assumed that the selection would be either a single row or a single column. Suppose you want the macro to check the selection. If it is a single row or column, you want to continue with the macro, but otherwise, you want to display an alert box and halt the macro. You can do so with the following macro:

The GOTO function branches to the Error macro in cell A13 if the logical test in the IF function is TRUE.



In the above Months macro, if the selection contains more than one range or if that range is not a single row or column, the GOTO function causes calculation to branch to cell A13 — the start of the Error macro. Excel continues to calculate down from that cell. When Excel encounters the RETURN function in cell A15, it stops the macro.

Branching with the ref Function

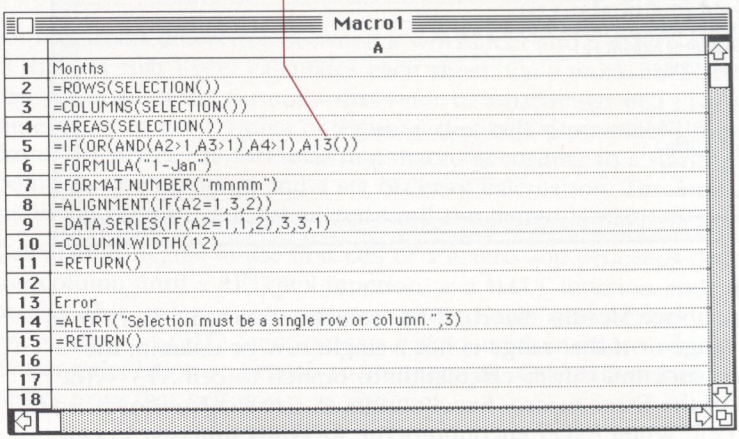
Branching with the ref function

Instead of using the GOTO function, it is possible to cause calculation to branch from one macro to another in such a way that when the second macro is done, calculation branches back to the first macro and continues in that macro. This is referred to as "calling" a macro from within another macro.

The form of a function that calls another macro is ref( ), where ref is a reference to another macro. When Excel encounters such a function, it branches to the upper-left corner cell in that reference. Calculation then proceeds down the column from that cell until a RETURN function is encountered. At that point, Excel branches back to the cell containing the ref( ) function and proceeds down the column from that cell until a RETURN function ends the macro.

For example, in the Months macro, if the selection is not a single column, you might want to bring up an alert box, but continue with the macro afterwards rather than end it. You could do so by calling the Error macro with the function A13( ) rather than branching to Error with GOTO(A13).

The A13( ) function calls the Error macro in cell A13 if the logical test in the IF function is TRUE.



In the above Months macro, if the selection contains more than one range or if that range is not a single row or column, the function A13( ) calls the macro at cell A13. Excel continues to calculate down from that cell. When Excel encounters the RETURN function in cell A15, it branches back to cell A5, the cell containing the function A13( ). Calculation then proceeds down the column from cell A5 until the RETURN function in cell A11 ends the macro.

Writing a command-equivalent macro

Writing a Command-Equivalent Macro

As an example of the power of macros, below is a macro that is the equivalent of the Create Names command from the Formula menu. It takes the text values in the left column of the selection and defines them as names for the rows in the selection.

This example demonstrates that an Excel command can be written as a macro. So, if you have ever thought of a command that you would like to have in Excel that isn't already there, you could probably write it as a macro.

Looping

The macro uses a technique known as "looping." It sets up a "counter" in cell A11. The formula of cell A11 is =A11+1. When the macro first begins, the value of cell A11 is set to 1 with the formula =SET.VALUE(A11,1). The first time through, the macro creates a name for the first row of the selection. Then, when Excel calculates the formula in A11, it increments the value of A11 by one. Excel then loops back in the macro and creates a name for the second row. Each time through the loop, the macro checks the value of the counter in A11 against the total number of rows in the selection. When it has created a name for all of the rows, the macro stops looping.

Macros

	A
1	Row.Names
2	=SET.NAME("Range",SELECTION())
3	=ROWS(Range)
4	=COLUMNS(Range)
5	=SET.VALUE(A11,1)
6	=IF(TYPE(INDEX(Range,A11,1))<>2,GOTO(A8))
7	=DEFINE.NAME(INDEX(Range,A11,1),INDEX(Range,A11,2):INDEX(Range,A11,A4))
8	=IF(NOT(ISERROR(A7)),GOTO(A11))
9	=ALERT("Illegal Name.",1)
10	=IF(A9=FALSE,HALT())
11	=A11+1
12	=IF(A11<=A3,GOTO(A6))
13	=RETURN()
14	
15	
16	
17	
18	

For more information on controlling calculation of macros, see "Macro Sheet Calculation" in Chapter 7, "More Information about Macro Sheets."

For further examples of command macros, see the Excel Data disk.

6 Function Macros

This chapter describes Excel's function macros. You'll learn how to enter, name, and use function macros, and you'll see some examples of typical ones. In addition, you'll find information on enhancing function macros and using the ARGUMENT and RESULT functions.

What Is a Function Macro?

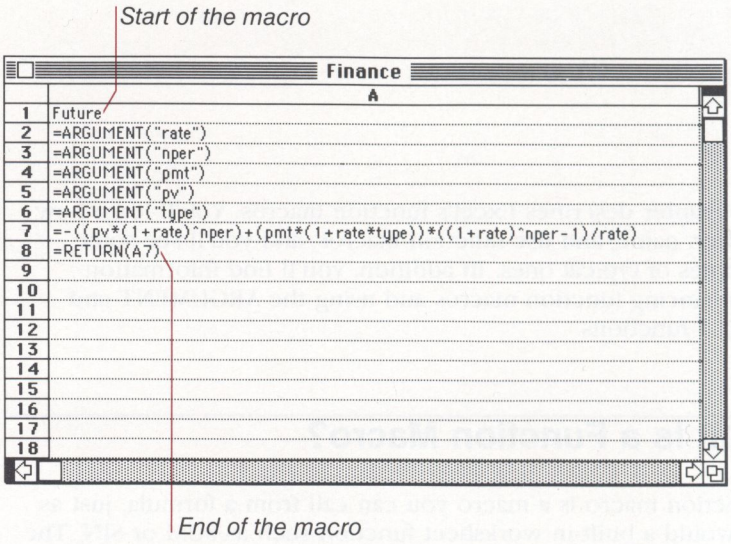
A function macro is a macro you can call from a formula, just as you would a built-in worksheet function such as SUM or SIN. The difference is that, because you create the macro, you decide what the function does. A function macro, like all macros, is made up of formulas that you enter in cells on a macro sheet. You cannot use the recorder to enter function macros.

For example, perhaps you regularly use a formula, or a set of formulas, that calculates the future value of an investment. You can enter these formulas on a macro sheet and give them a name such as Future. Then, you can use Future as a function on your worksheet, without having to write out repeatedly the formulas that comprise Future.

Function macros are another way in which Excel simplifies the creation of worksheets. Once you have entered a complex or often-used calculation as a function macro, all you have to remember are its name and arguments.

Any formula or set of formulas that you enter on your worksheets can be written as a function macro on a macro sheet. Often you need only two Macro Language functions — ARGUMENT and RETURN — to create a function macro from a formula.

Here's an example of what a function macro looks like:



After you become more familiar with the Excel Macro Language, you may want to write more elaborate function macros.

There are three steps to working with function macros: entering, naming, and using them in formulas.

Entering a function macro

Entering a Function Macro

To enter a function macro, you enter formulas in cells on a macro sheet. Begin with the formulas that contain the RESULT and ARGUMENT functions. Then enter the formulas that carry out the function macro's calculation. End the function macro with a formula that contains the RETURN function, which returns the result of the macro calculation.

When entering a function macro, follow the downward order of calculation, as described in the section "Macro Sheet Calculation" in Chapter 7, "More Information about Macro Sheets."

Naming a Function Macro

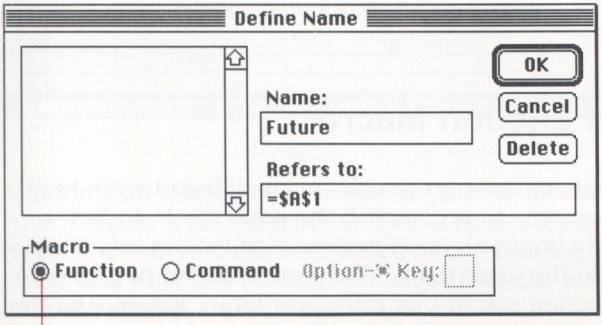
Before you use a function macro in a formula on a worksheet, you should define a name for it.

To name a function macro:

To name a function macro

1. Select the first cell of the function macro on the macro sheet.
2. Choose Define Name from the Formula menu. When a macro sheet is active, Define Name has some extra options, as shown below.
3. Type a name for the macro in the Name box.
4. Click Function.
5. Click OK.

The Define Name dialog box looks like this:



The Function button indicates that the named cell is at the start of a function macro.

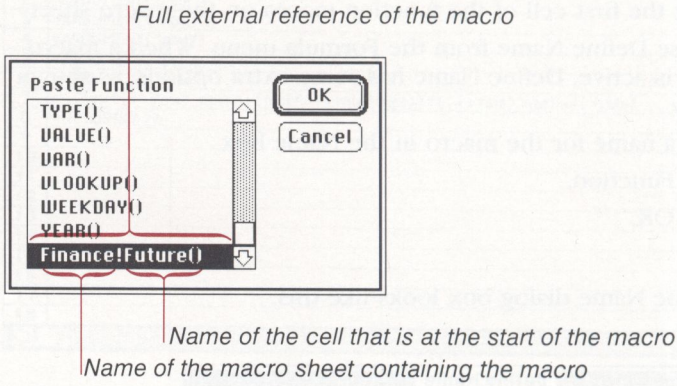
Using a Function Macro

Using a function macro

To use a function macro in a worksheet formula, you need to provide Excel with the same information as for a built-in function. That is, you must give the name of the macro, which is the external reference of the cell that starts the macro, followed by a left parenthesis, arguments separated by commas, and a right parenthesis.

There are two ways to enter a function macro in a formula; these procedures are the same as those for using worksheet functions. That is, you can use the Paste Function command from the Formula menu, or you can type it.

The Paste Function dialog box looks like this:



A Sample Function Macro

As with any program, there are usually many different ways to write a function macro to accomplish the same result. In fact, a good way to learn about function macros is to look at several different ways to do the same thing. This section shows two ways to write a basic function macro that calculates future values. And the following section illustrates a further enhancement to the macro.

The future value formula

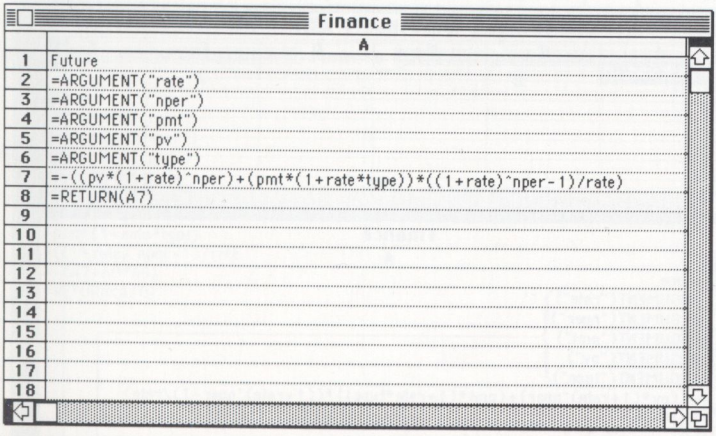
Excel has a built-in function, FV, to compute the future value of an investment involving constant cash flows. The arguments to FV are the interest rate per period (*rate*), the number of periods (*nper*), the periodic payment (*pmt*), the present value (*pv*), and an argument (*type*) specifying whether payments occur at the beginning or the end of the periods.

The FV function looks like this: FV(*rate,nper,pmt,pv,type*). The formula Excel uses for FV is the following:

$$pv*(1+rate)^{nper}+pmt*(1+rate*type)*\frac{(1+rate)^{nper}-1}{rate}+fv=0$$

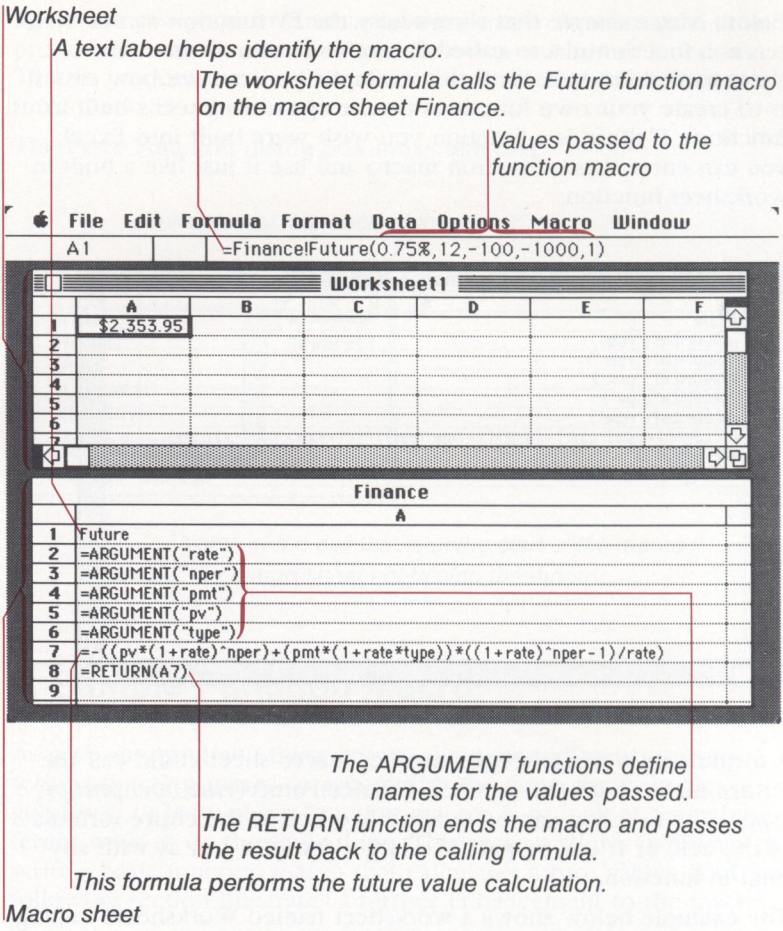
$$(for\ \ rate<>0)$$

Below is an example that shows how the FV function can be written as a function macro called Future and entered on a macro sheet named Finance. With this example, you can see how easy it is to create your own functions that are just like Excel's built-in functions. If there is a function you wish were built into Excel, you can enter it as a function macro and use it just like a built-in worksheet function.



A formula on a worksheet or another macro sheet could call the Future macro with the function Finance!Future(*rate,nper,pmt,pv, type*). The function can be entered in a cell as the entire formula of the cell, or it can be part of a larger formula, just as with any built-in function.

The example below shows a worksheet named Worksheet1 calling the Future function macro on the macro sheet named Finance.



When Excel calculates the formula in cell A1 on the worksheet named Worksheet1, it encounters the Finance!Future function. It then goes to the cell named Future on the macro sheet named Finance and starts calculating the macro there. Each ARGUMENT function on the macro sheet takes one argument from the formula on the worksheet and defines a name for it. So, the first formula =ARGUMENT("rate") takes the value of the first argument, 0.0075, and defines the name rate for it. Similarly, the names nper, pmt, pv, and type are defined for the values 12, −100, −1000, and 1.

The formula in cell A7 on the macro sheet computes the future value, referring to the values of the arguments by name. When Excel calculates the RETURN function, it stops the macro, returns the value computed in cell A7 to the Finance!Future function on the worksheet, and then continues calculating the worksheet.

The next example shows a slightly different version of the Future macro. In this case, the formula is broken up into several smaller formulas.

Finance

	A	B
1	Future	
2	=ARGUMENT("rate")	
3	=ARGUMENT("nper")	
4	=ARGUMENT("pmt")	
5	=ARGUMENT("pv")	
6	=ARGUMENT("type")	
7	=pv*(1+rate)^nper	
8	=pmt*(1+rate*type)	
9	=((1+rate)^nper-1)/rate	
10	=-(A7+A8*A9)	
11	=RETURN(A10)	
12		
13		
14		
15		
16		
17		
18		

Future shows you how easy it is to write a function macro. Like Future, the majority of function macros require only two Macro Language functions, ARGUMENT and RETURN.

Enhancing Function Macros

Excel's built-in worksheet function FV does several things that the sample function macro, Future, does not. For example, if the *rate* argument is 0, the formula in cell A9 of the Future macro shows the error value #DIV/0!. The built-in FV function instead uses the formula

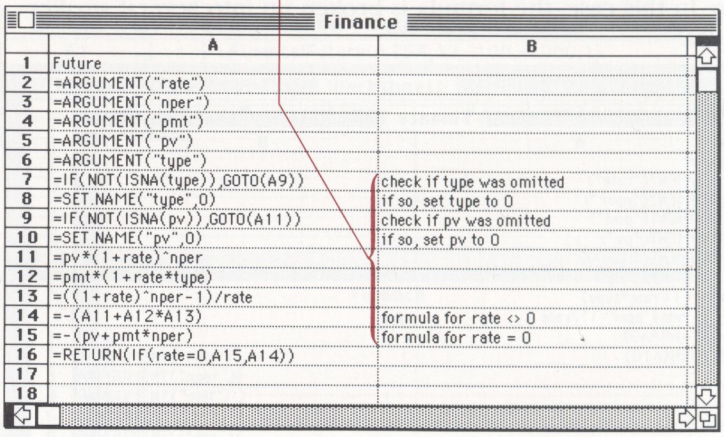
$$pv + pmt * nper + fv = 0$$

to calculate the function. Also, in the built-in function, if you omit either the *type* or *pv* argument, it is assumed to be zero.

Enhancing the Future macro

You can add these enhancements to the Future function macro as follows:

Comments are included to make the macro easier to understand.



Finance

	A	B
1	Future	
2	=ARGUMENT("rate")	
3	=ARGUMENT("nper")	
4	=ARGUMENT("pmt")	
5	=ARGUMENT("pv")	
6	=ARGUMENT("type")	
7	=IF(NOT(ISNA(type)),GOTO(A9))	check if type was omitted
8	=SET.NAME("type",0)	if so, set type to 0
9	=IF(NOT(ISNA(pv)),GOTO(A11))	check if pv was omitted
10	=SET.NAME("pv",0)	if so, set pv to 0
11	=pv*(1+rate)^nper	
12	=pmt*(1+rate*type)	
13	=((1+rate)^nper-1)/rate	
14	=-(A11+A12*A13)	formula for rate <> 0
15	=-(pv+pmt*nper)	formula for rate = 0
16	=RETURN(IF(rate=0,A15,A14))	
17		
18		

If you omit either the *type* or *pv* argument in the Finance!Future function, Excel automatically uses the error value #N/A as the value of the argument. However, if the names type and pv have the value #N/A, then all the formulas that use those names will also have the value #N/A. The function macro above checks the values of the names type and pv and, if they are #N/A, changes them to 0 using the SET.NAME function.

The SET.NAME function is particularly useful in function macros. For more information on this function, see Chapter 8, "Macro Language Function Directory."

Notice the comments in the above example. These can be entered as constant text values in cells adjacent to the actual macro to help you follow the actions of the macro. For more information, see "Using Labels and Comments in Macros" in Chapter 7, "More Information about Macro Sheets."

The ARGUMENT and RESULT Functions

Using ARGUMENT to define names and to specify argument types

In all the examples of function macros shown above, the ARGUMENT function was used to define a name, such as rate or nper, for the value of that argument. Those arguments were then referred to by name in other formulas in the macro. The ARGUMENT function has some additional capabilities as well.

For example, you can use the ARGUMENT function to indicate that the arguments must be of a certain type, such as a number, text, or logical value. Excel checks to see if the value passed to the function macro is of the specified type. If it is not, Excel first attempts to convert it to the specified type. If the value cannot be converted, the macro returns the error value #VALUE! without running.

Using RESULT to specify result types

Excel has another function like the ARGUMENT function, called RESULT. With the RESULT function, you can indicate that the result returned by a function macro must be of a certain type. Again, if it is not, Excel first attempts to convert it to the specified type. If the value cannot be converted, the macro returns the error value #VALUE!.

For example, in the Future macro (see "A Sample Function Macro"), the arguments *rate*, *nper*, *pmt*, *pv*, and *type* should all be numbers, and the result of the function should be numbers. In the example below, the number 1 in the ARGUMENT and RESULT functions tells Excel that the arguments and the result of the function macro are single numbers and not text or other values. For a complete description of what numbers to use in the ARGUMENT and RESULT functions to specify different types, see "Argument Types" and "Result Types" in Chapter 7, "More Information about Macro Sheets."

Finance

	A	B
1	Future	
2	=RESULT(1)	
3	=ARGUMENT("rate",1)	
4	=ARGUMENT("nper",1)	
5	=ARGUMENT("pmt",1)	
6	=ARGUMENT("pv",1)	
7	=ARGUMENT("type",1)	
8	=IF(NOT(ISNA(type)),GOTO(A10))	check if type was omitted
9	=SET.NAME("type",0)	if so, set type to 0
10	=IF(NOT(ISNA(pv)),GOTO(A12))	check if pv was omitted
11	=SET.NAME("pv",0)	if so, set pv to 0
12	=pv*(1+rate)^nper	
13	=pmt*(1+rate*type)	
14	=((1+rate)^nper-1)/rate	
15	=-(A12+A13*A14)	formula for rate <> 0
16	=-(pv+pmt*nper)	formula for rate = 0
17	=RETURN(IF(rate=0,A16,A15))	
18		

If an argument to your function macro is supposed to be a number, text, or logical value, or a reference to a cell containing a number, text, or logical value, then you need not specify a type in the ARGUMENT function. Similarly, if your function macro is supposed to give a number, text, or logical value as a result, you may omit the RESULT function entirely. This was illustrated in the examples of the Future function macro in "A Sample Function Macro."

Using ARGUMENT to enter values

You can also use the ARGUMENT function to enter the values of the arguments in cells on the macro sheet, instead of to define names for them. Then, you can refer to those cells in other formulas in the macro.

In the example below, the first ARGUMENT function, ARGUMENT("rate",1,A18), tells Excel to enter the value of the first argument passed in cell A18. It also tells Excel to define the name rate to refer to the cell A18, rather than to the value itself. Notice that because the values are entered in cells rather than assigned to names, SET.VALUE functions, rather than SET.NAME functions, are used to change the values.

Finance

	A	B
1	Future	
2	=RESULT(1)	
3	=ARGUMENT("rate",1,A18)	store in A18
4	=ARGUMENT("nper",1,A19)	store in A19
5	=ARGUMENT("pmt",1,A20)	store in A20
6	=ARGUMENT("pv",1,A21)	store in A21
7	=ARGUMENT("type",1,A22)	store in A22
8	=IF(NOT(ISNA(type)),GOTO(A10))	check if type was omitted
9	=SET.VALUE(type,0)	if so, set type to 0
10	=IF(NOT(ISNA(pv)),GOTO(A12))	check if pv was omitted
11	=SET.VALUE(pv,0)	if so, set pv to 0
12	=pv*(1+rate)^nper	
13	=pmt*(1+rate*type)	
14	=((1+rate)^nper-1)/rate	
15	=-(A12+A13*A14)	formula for rate <> 0
16	=-(pv+pmt*nper)	formula for rate = 0
17	=RETURN(IF(rate=0,A16,A15))	
18	0.75	

Using Arrays in Function Macros

Sometimes you want to create functions that accept and/or return arrays rather than single values.

For example, you can write a function macro that calculates the depreciation of an asset over the expected life of the asset, using the declining balance method. The arguments to the function would be the starting book value of the asset (*sbv*), the salvage value (*sal*), the useful life expectancy (*n*), and the declining-balance factor (*i*) expressed as a percentage. The function would produce an array of depreciation values; each value in the array would be the depreciation in one year in the life of the asset.

Suppose, for example, a machine you bought for $10,000 has an expected life of 5 years and a salvage value of $500. Using a 200% declining balance, the depreciation of the asset over 5 years would be given by the function Finance!Dep(10000,500,5,200%). If this function were entered as an array formula in cells A1:A5, it would produce the depreciation values $4,000, $2,400, $1,440, $864, and $796 in those cells.

The Dep function macro

If you name your depreciation function macro Dep, and enter it on a macro sheet called Finance, it could look like this:

Finance

	A	B
1	Dep	
2	=RESULT(64)	
3	=ARGUMENT("sbv",1)	starting book value
4	=ARGUMENT("sal",1)	salvage value
5	=ARGUMENT("n",1)	life expectancy
6	=ARGUMENT("i",1)	declining-balance factor
7	=SET.VALUE(rbv,sbv)	
8	=SET.VALUE(k,1)	
9	=rbv*i/n	A9 is named dpn; start of loop
10	=IF(rbv-dpn>=sal,dpn,rbv-sal)	A10 is named dpn2; check if < sal
11	=SET.VALUE(INDEX(results,k),dpn2)	
12	=rbv-dpn	A12 is named rbv (remaining value)
13	=k+1	A13 is named k; increment k
14	=IF(k<=n,GOTO(dpn))	end of loop
15	=RETURN(INDEX(results,1):INDEX(results,n))	return n values from results
16	4000	A16:A25 is named results
17	2400	
18	1440	

The Dep function macro loops from cell A14 back to cell A9, once for each year in the life expectancy of the asset. Each time through the loop, the function macro calculates a depreciation value and enters it in one of the cells in the result range A16:A25. When it is finished looping, the function macro returns the n by 1 array of depreciation values entered in an n by 1 range of cells named results.

In a function macro that returns an array, it is very common to enter the values into a range of cells on a macro sheet and then return that array of values with the RETURN function. In the above function macro, the range of cells into which the depreciation values are entered is ten rows high by one column wide. That means that the Dep function, as written, limits you to depreciating the asset over a life expectancy of 10 years. If n is larger than the size of results, the function will produce an error value. You can remove this limitation by using the OFFSET function instead of the INDEX function. For more information on the OFFSET function, see Chapter 8, "Macro Language Function Directory."

For further examples of function macros, see the Excel Data disk.

7 More Information about Macro Sheets

To help you use and understand macros, this chapter gives some additional information about how macros and macro sheets work.

You'll learn about using Excel's recorder, organizing macro sheets, using labels and comments, and calling macros. In addition, you'll find information on how Excel calculates macro sheets and how it passes arguments and results.

Setting the Recorder Range for a Command Macro

Before you begin recording a new command macro, always make a new selection on the macro sheet and carry out the Set Recorder command. The Set Recorder command designates your selection as the "recorder range." The next time you choose Start Recorder, your actions will be recorded in this range.

Selecting the recorder range

There are two ways to select the recorder range. You can select one cell, or you can select a range of cells. If you select one cell, Excel uses the entire column, from that cell down, as the recorder range. If you select a range of cells, Excel uses that range as the recorder range.

When making the selection, be sure you leave enough room for the recording you want to make. Each action you take during recording, such as entering a formula, making a new selection, or carrying out a command, will take up one cell in the recorder range.

Starting, Stopping, and Restarting the Recorder

When you choose Start Recorder, Excel finds the uppermost blank cell in the recorder range — by searching upward from the bottom of the range — and starts recording there. If your recorder range is all or part of a single column, recording proceeds down the column. If your recorder range is a range of cells more than one column wide, recording proceeds down the first column of the range, then down the second, and so on. Excel automatically enters a GOTO function in the last cell of each column in the range to direct calculation to the top of the next column.

When you choose Stop Recorder, Excel automatically enters a RETURN function as the last formula of the macro. When the macro is run or called, the RETURN function ends the macro. If you choose Start Recorder again, without resetting the recorder range, Excel writes over the RETURN function with your next action and continues recording.

In this way, the Excel recorder is similar to a tape recorder: if you stop the tape and start it again, recording continues where you left off. Thus, as long as there are blank cells in the recorder range, you can stop in the middle of recording a macro, make changes to your documents, carry out commands, or do nothing at all, and then start the recorder again, without resetting it. If you are starting a new macro, however, you should start with a new recorder range.

What to do if you get "Recorder range full"

If the recorder reaches the end of the recorder range, Excel displays an alert box that says "Recorder range full" and stops recording. If this happens, and you want to continue recording your macro, you must extend the recorder range. The procedure is the same as setting the recorder range for a new macro, but the two recordings must be connected together.

If there is room on the macro sheet, set a new recorder range that includes the old recorder range but extends beyond it. If there is not room, you can set a new recorder range that is separate from the old recorder range. Then you must replace the RETURN function at the end of the old recorder range with a GOTO function that branches to the start of the new recorder range.

Using Relative Record and Absolute Record

When you record a command macro, you can choose between two settings for the recorder: Relative Record and Absolute Record. Excel normally uses Relative Record, but you can change back and forth between the two with the Relative/Absolute Record command from the Macro menu.

Using Relative Record

The difference between Relative Record and Absolute Record becomes important when you select cells while recording. With Relative Record on, the action of selecting cells is recorded with relative references. The references are relative to the cell that was active before the selection was made. When you run the macro, it selects cells relative to that cell, although that cell may be different from the cell that was active when you recorded the macro.

For example, suppose you are recording a macro with Relative Record on and the active cell on the worksheet is A1. When you select cells A1:L1, that action is recorded using relative references. Later, when you run the macro, if C3 was the active cell before the selection was made, the macro will select cells C3:N3.

Using Absolute Record

With Absolute Record on, the action of selecting cells is recorded with absolute references. When you run the macro, it selects the very same cells you selected while recording.

For example, suppose you recorded the above macro with Absolute Record on. Later, when you run the macro, if C3 (or any other cell) was active before the selection was made, the macro will still select the cells A1:L1.

You can switch between Relative Record and Absolute Record any time you wish, even in the middle of recording a macro, by choosing the Relative Record or Absolute Record command from the Macro menu. When you first start Excel, Relative Record is on. If you want to record the next selection using absolute references, choose the Absolute Record command and then make the selection. Excel will record all selections thereafter with absolute references until you choose Relative Record.

Referencing the Active Worksheet

Occasionally, in a formula on a macro sheet, you may want to refer to a cell, a range of cells, or a name on the active worksheet. Because the active worksheet is different from the macro sheet containing the formula, the reference must be an external

reference. However, you don't want the reference to contain the name of a specific worksheet. Instead, you want it always to refer to the active worksheet, which may be different each time you run the macro. The syntax for an external reference to the active worksheet is the same as that for a normal external reference, but the worksheet name is omitted.

For example, the references !$A$1, !$C$3:$E$5, and !Sales refer to the cell A1, the range of cells C3:E5, and the name Sales on the active worksheet. The exclamation point indicates that it is an external reference, rather than a reference to the macro sheet.

Organizing Your Macro Sheets

If you have used macros with other spreadsheet products, you probably put the macros right on the worksheet — placed in some far corner of the sheet. This took up a lot of memory. And if you wanted to use the same macros on different worksheets, you had to copy the macros to those worksheets.

You will find that Excel's macro sheets are a much more convenient and efficient medium for writing and storing macros. The same macro sheet can be used with all your worksheets, so there is no need to copy macros onto different sheets.

With Excel, you can have many command macros and function macros on the same macro sheet. In fact, if you have a collection of macros that you use frequently, it is wise to have them all on one sheet. That way, you save memory by reducing the number of sheets that must be open at one time. If you have some macros that you use only occasionally with certain worksheets, you might want to put them on a separate macro sheet and load it only when you need it.

The amount of memory a sheet takes up is determined by the lowermost row and rightmost column containing a formatted or non-blank cell. There is no need to leave large blank areas between macros on a macro sheet. You save memory if the macros are close together.

Deleting a macro

Remember that the cells and cell contents on a macro sheet behave exactly like those on a worksheet. Deleting a macro from a macro sheet is the same as deleting or clearing cells from a worksheet. Deleting the name of a macro is the same as deleting any Excel name. For more information, see Chapter 5, "Editing a Worksheet," in the *User's Guide*.

Using Labels and Comments in Macros

You can enter a constant value in any cell of a macro without affecting the behavior of the macro. When a macro encounters a constant value during calculation, it skips over it and continues calculating in the next cell. Therefore, you can use constant text values as labels and comments to document your macros.

Using labels in macros

It is particularly useful to enter the name of a macro as text in the first cell of the macro. This makes it easy to identify macros when looking at a macro sheet. The name should be descriptive of what the macro does. Also, when you name the macro by selecting the first cell of the macro and choosing the Define Name command, this text will be the proposed response in the Name box of the dialog box.

It is a good practice to use labels to identify key points in a macro, just as you would use labels in a program. It is especially recommended that you use labels for all GOTO function destinations.

To label a GOTO function destination

To label a GOTO function destination:

1. Enter text in the destination cell
2. Using the Define Name command, name the cell, using that text as the name.
3. Use the name of the cell, rather than the cell reference, in the GOTO function.

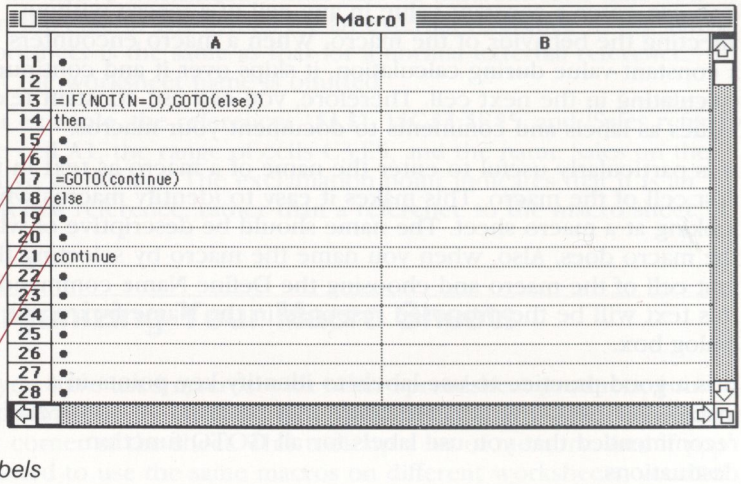
It is much easier to follow the logical flow of a macro if each GOTO destination has a label.

For example, the following macro:

Macro1

	A	B
11	•	
12	•	
13	=IF(NOT(N=0),GOTO(A17))	
14	•	
15	•	
16	=GOTO(A19)	
17	•	
18	•	
19	•	
20	•	
21	•	
22	•	
23	•	
24	•	
25	•	
26	•	
27	•	
28	•	

is better written as:



where cells A14, A18, and A21 are named then, else, and continue, respectively.

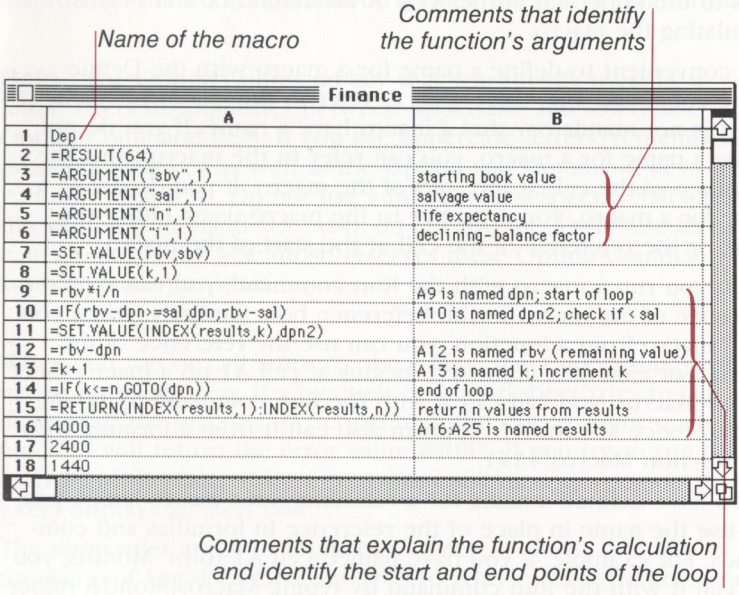
When you enter text in a cell and use that text as the name of the cell, as in the above example, you should enter text that is a legal Excel name. That is, it should consist only of letters, numbers, periods (.), and underlines (_). For more information, see "Names" in Chapter 4 of the *User's Guide*.

Note When you name a cell for labeling purposes, be careful not to use a name that you have used in an ARGUMENT function on the same macro sheet. If you do, the name will be changed the next time the ARGUMENT function is calculated. (See also "Defining Names for Arguments" in this chapter.)

Using comments in macros

You can document your macros even more fully by adding comments. Again, comments are simply constant text values in cells. Usually, you will enter your comments in a column adjacent to the macro. Use comments to describe the calculation or action that is taking place at each point in the macro.

The example below illustrates a well-documented macro.


Name of the macro
Comments that identify the function's arguments
Finance
A
B
1 Dep
2 =RESULT(64)
3 =ARGUMENT("sbv",1) starting book value
4 =ARGUMENT("sal",1) salvage value
5 =ARGUMENT("n",1) life expectancy
6 =ARGUMENT("i",1) declining-balance factor
7 =SET.VALUE(rbv,sbv)
8 =SET.VALUE(k,1)
9 =rbv*i/n A9 is named dpn; start of loop
10 =IF(rbv-dpn>=sal,dpn,rbv-sal) A10 is named dpn2; check if < sal
11 =SET.VALUE(INDEX(results,k),dpn2)
12 =rbv-dpn A12 is named rbv (remaining value)
13 =k+1 A13 is named k; increment k
14 =IF(k<=n,GOTO(dpn)) end of loop
15 =RETURN(INDEX(results,1):INDEX(results,n)) return n values from results
16 4000 A16:A25 is named results
17 2400
18 1440
Comments that explain the function's calculation and identify the start and end points of the loop

For further examples of macros that have labels and comments, see the sample macros on the Excel Data disk.

Calling and Running Macros

Three ways to call a macro

There are three ways that a macro can be called.

- By the user with the Run command (or with an Option-Command-key, which is the same as using the Run command, but quicker)—for command macros only
- By a function in a worksheet formula when the formula is calculated—for function macros only
- By a function in a macro sheet formula when the formula is calculated—for command and function macros

In all cases, you must provide Excel with a reference to the macro. This reference will always be an external reference, except when a formula on a macro sheet calls another macro on the same sheet. An external reference consists of the name of the macro sheet, followed by an exclamation point (!), followed by

the reference or name of the start of the macro, such as Macros!$A$1 or Macros!Months. When a macro is called, Excel goes to the upper-left corner cell of the reference and begins calculating the macro.

It is convenient to define a name for a macro with the Define Name command and, thereafter, to refer to the macro by name. But it is not mandatory that a macro have a name. If you do not define a name for a macro, you can refer to the macro using the reference of the cell at the start of the macro. If you do define a name for a macro, you can refer to the macro using either the name or the reference of the cell at the start of the macro.

When you run a macro with the Run command, you can type the reference of the macro in the Reference box. Similarly, when you call a macro from a function, you can use the reference of the macro. For example, if a macro begins at cell A1 on a macro sheet named Macros, you can call it with the Run command by typing the reference Macros!A1. You can also call it from a formula with the function Macros!$A$1.

If you have defined a name for a reference on a macro sheet, you can use the name in place of the reference in formulas and commands. For example, if you have named cell A1 to be Months, you can call it with the Run command by typing Macros!Months rather than Macros!A1. You can also call it from a formula with the function Macros!Months( ) rather than Macros!$A$1. Because Excel begins calculating at the upper-left corner cell of a macro, it is only necessary to name that cell; you do not need to name the entire macro.

Only when you call a macro from a formula that is on the same macro sheet can you use an on-sheet reference rather than an external reference. For example, a formula on the macro sheet named Macros can contain the function Months( ) or $A$1 that calls a macro named Months that begins at cell A1 on the same macro sheet. Because the macro being called is on the same sheet as the calling macro, it is not necessary to use an external reference.

Using the Define Name command

On a macro sheet, the dialog box of the Define Name command has three options that are not found when the command is used on a worksheet: a Command button, a Function button, and a box for typing an Option-Command-key. These options are there only for your convenience. If you click the Function button, the name you designate will appear in the list box in the Paste Function command, so you can choose the name instead of typing it in. If you click the Command button, that name will appear in the list box in the Run command, so you can choose the name instead of typing it in. When you click the Command button, you'll also be able to specify an Option-Command key code, so that you can run

the macro by typing that key while pressing the Option and Command keys. If you do not use these options, you can still run the macro with the Run command or include the macro in a formula.

Note The Option-Command-key that you specify must be a letter. Excel distinguishes between uppercase and lowercase Option-Command-keys. If you use uppercase letters, remember to press the Shift key as well as the Option and Command keys when you call the macro. Do not use a number or a special character as the Option-Command-key. Also, do not use the lowercase letters e, u, i, or n, as the Macintosh reserves these Option-Command-keys for other purposes.

When you run or call a macro, the macro sheet containing that macro must be open. If a function on a worksheet tries to call a macro and the macro sheet containing the macro is not open, the function will return the error value #REF!. If you try to run a macro and the macro sheet containing the macro is not open, Excel displays an alert box.

The same rules apply to the names of macro sheets as apply to the name of any document referred to with an external reference. That is, the name of a macro sheet should be a legal Excel name, consisting of letters, digits, periods (.), and underlines (_). If the name of the macro sheet is not a legal Excel name, you must enclose it in single quotation marks, for example 'Macros-1'!$A$1 or 'Statistical Functions'!Distribution. For a complete definition of legal Excel names, see "Names" in Chapter 4 of the *User's Guide*.

Macro Sheet Calculation

An important difference between a worksheet and a macro sheet is the way Excel calculates each sheet. On a worksheet, Excel calculates all the formulas that refer to a particular cell every time you edit the formula or value in that cell. On a macro sheet, Excel calculates the formulas in a macro according to a starting point, an ending point, and a set order of calculation. The formulas in a macro are not calculated until the macro is run or called.

To explain this difference and help you use macro sheets as efficiently as possible, this section covers four main topics. The first part discusses the basic downward calculation of macros, and the concepts of branching to and calling macros. The next part explains formulas with action-taking functions. The third part tells you how to control macro calculation with looping and conditionals, and the last part describes how to interrupt a macro.

How Excel Calculates Macros

Macro calculation begins in the first cell of a macro and proceeds down the column until a RETURN function is encountered. If you run a macro with the Run command (or an Option-Command-key), the RETURN function ends the macro. If the macro is called by a function during the calculation of a worksheet or macro sheet formula, the RETURN function ends the macro and can return values to the formula. Then, Excel continues its calculation of formulas on the worksheet or macro sheet.

Contents of a cell in a macro

In a macro, a cell can contain any one of the following:

- a formula that produces a value
- a formula that takes an action
- a constant value

If a cell contains a formula that produces a value, but does not contain action-taking functions, Excel calculates the formula and enters the value in the cell.

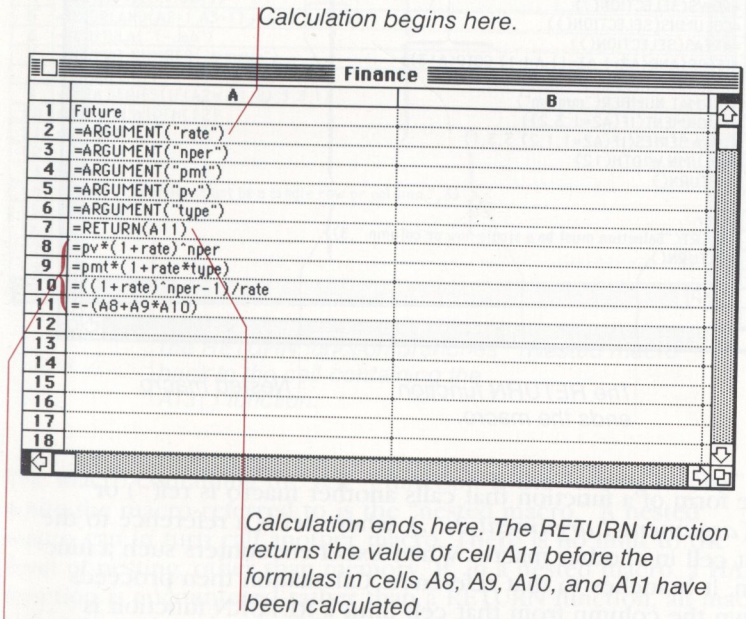
If a cell contains a formula with an action-taking function, the action is taken when Excel calculates the formula (see "Formulas with Action-Taking Functions" below).

If a cell contains a constant value, such as a number or a text value, Excel skips over it to the next cell. Therefore, you can enter constant values anywhere in a macro without affecting the behavior of the macro (see "Using Labels and Comments in Macros" above).

A formula in a macro may be an array formula that is entered into one or more cells, although this is not very common. In calculating a macro, if Excel encounters the upper-left corner of a range of cells containing an array formula, it calculates the array formula just as it would any other formula. It then continues calculation below the lower-left corner of the array formula. If Excel encounters a cell in an array formula that is not the upper-left corner cell, it does not calculate the formula, but continues calculation below the array formula.

For example, suppose that cells C3:E5 on a macro sheet contain an array formula and that Excel is currently calculating cell C2. Next, it goes to cell C3 and calculates the array formula in all of cells C3:E5. Then, it continues calculation at cell C6. Were Excel currently calculating D2, however, it would skip over the array formula and continue calculation at cell D6.

Remember that the formulas in a macro are calculated only when the macro is run or called. Therefore, the formulas on a macro sheet must be in the path of a macro to be calculated. For example, the following macro will not work properly because the three formulas in cells A8, A9, and A10 are never calculated.

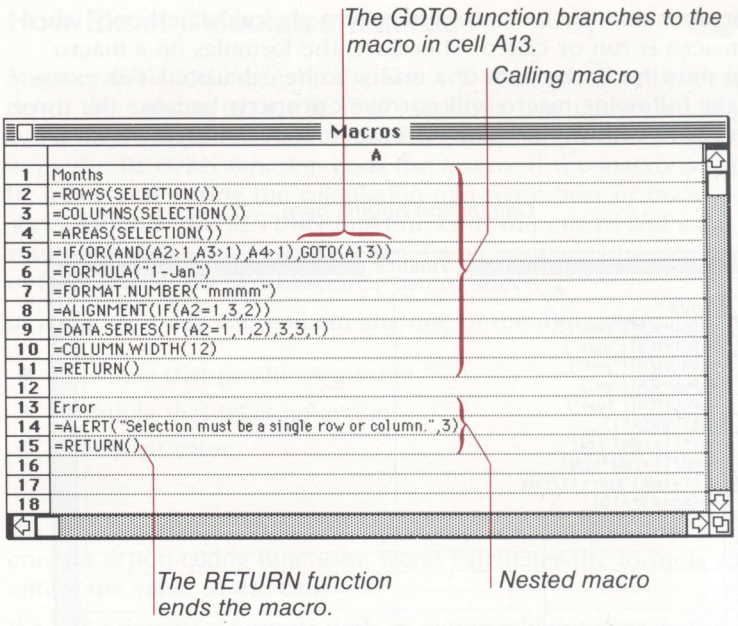


Branching to another macro

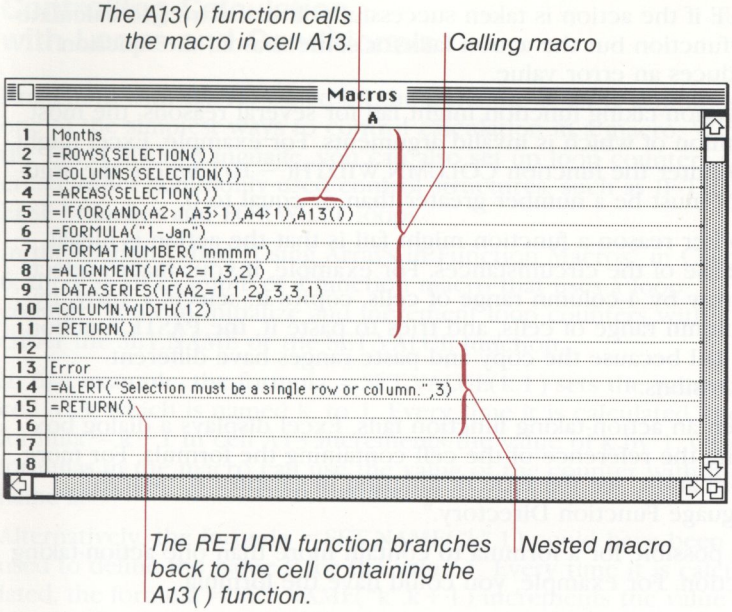
The order of calculation in a macro is altered only when Excel encounters either a GOTO function or a function that calls another macro.

These functions cause calculation to continue at another cell. This is called "branching."

The form of the GOTO function is GOTO(*ref*), where *ref* is a reference to cells on a macro sheet. When Excel encounters a GOTO function, it branches to the upper-left corner cell of *ref*. Calculation then proceeds down the column from that cell until a RETURN function ends the macro.



The form of a function that calls another macro is ref( ) or ref(*argument-1,argument-2,...*), where ref is a reference to the first cell in another macro. When Excel encounters such a function, it branches to that reference. Calculation then proceeds down the column from that cell until a RETURN function is encountered. When the RETURN function is encountered, calculation branches back to the cell containing the ref( ) function and proceeds down the column from that cell.



The macro containing the ref( ) function is the "calling macro," while the macro referred to is the "nested macro." A nested macro can in turn call another macro. There is no limit to the level of nesting, other than memory. If, in a nested macro, a HALT function is encountered rather than a RETURN function, all macros end, not just the nested macro.

In general, a RETURN function should end a macro. A HALT function should only be used in a macro when you want to end not only that macro but any macros that may have called that macro. You may want to do this, for example, if an error has occurred.

Formulas with Action-Taking Functions

When a formula in a macro contains an action-taking function, the action is carried out when the formula is calculated. Although you are primarily interested in the actions they take, these formulas also return values, as does every other formula and function in Excel.

Values returned by action-taking functions

Almost all action-taking functions produce either the value TRUE, the value FALSE, or an error value.

When you first enter an action-taking function (either by recording or typing), Excel does not actually calculate the function. Its value is FALSE, indicating that the action has not been taken. When an action-taking function is calculated, it produces the value

TRUE if the action is taken successfully. If Excel tries to calculate the function but the action fails for some reason, the function produces an error value.

Why an action-taking function might fail

An action-taking function might fail for several reasons, the most common of which is invalid arguments. For example, Excel might encounter the function COLUMN.WIDTH(−10), where the argument must be a number greater than or equal to 0.

Another reason a function might fail is that the action is illegal because of the circumstances. For example, if your macro selects a 3-row by 4-column range of cells, copies it, selects a 4-row by 3-column range of cells, and tries to paste it, the PASTE function will fail because the copy and paste ranges have different dimensions.

When an action-taking function fails, Excel displays a dialog box giving the reference of the cell containing the formula. For more information, see the ERROR function in Chapter 8, "Macro Language Function Directory."

Formulas with more than one action-taking function

It is possible for a formula to contain more than one action-taking function. For example, you could have the formula

```
=COLUMN.WIDTH(10)
   +ALIGNMENT(3)*FORMAT.NUMBER("mmmm")
```

When Excel calculates this formula from left to right, all three actions will be taken and all three functions will produce the value TRUE. The value of TRUE+TRUE *TRUE is 2, so the formula will produce the value 2. However, in general, an action-taking function should always appear in a formula by itself; there is no advantage to combining action-taking functions in a formula, and the results can be unpredictable.

Note that the INPUT and ALERT functions are not action-taking functions, so they do not follow the above rules. (For a complete description of these functions, see Chapter 8, "Macro Language Function Directory.") The INPUT function is often used in the same formula with real action-taking functions. For example, you might use the formula

```
=FORMAT.NUMBER(INPUT("Enter a date format:",2))
```

When Excel calculates this formula, it first calculates the INPUT function, because it is an argument to the FORMAT.NUMBER function. Then, Excel displays the INPUT dialog box and waits for your input. After you type the input, the INPUT function gives the input as its value. Excel then calculates the FORMAT.NUMBER function using the input as its argument.

Controlling Calculation with Loops and Conditionals

Branching with GOTO based on the outcome of an IF condition is one of the simplest ways to control calculation in a macro. With the Excel Macro Language, you can also set up loop counters and create other commonly-used control structures, such as IF-THEN-ELSE branches and FOR-NEXT loops.

Setting up loop counters

In the section called "Using Arrays in Function Macros" in Chapter 6, there is an example of a function macro that uses a loop counter. You can initialize and increment loop counters with either the SET.NAME or the SET.VALUE function.

In that example, the formula =SET.VALUE(k,1) sets the value of cell A13, which is named k, to 1. Every time it is calculated, the formula =k+1 in cell A13 increments the value of k by 1. Other formulas in the macro can use the value of the counter with the name k.

Alternatively, the formula =SET.NAME("k",1) could have been used to define the name k for the value 1. Every time it is calculated, the formula =SET.NAME("k",k+1) increments the value of k by 1.

The formula =SET.NAME("k",k+1) or =k+1 must be in the path of the macro, or else the counter will not be incremented.

Common conditional programming structures

With Excel, you can set up various conditional programming structures to control the calculation of a macro.

You can set up an IF-THEN-ELSE condition in the following manner:

Macro1

	A	B
11	•	
12	•	
13	=IF(NOT(N=0),GOTO(else))	
14	then	
15	•	
16	•	
17	=GOTO(continue)	
18	else	
19	•	
20	•	
21	continue	
22	•	
23	•	
24	•	
25	•	
26	•	
27	•	
28	•	

You can set up a FOR-NEXT loop in the following manner:

Macro1

	A	B
11	•	
12	•	
13	=SET.VALUE(A17,1)	
14	begin	
15	•	
16	•	
17	=A17+1	
18	=IF(A17<=N,GOTO(begin))	
19	continue	
20	•	
21	•	
22	•	
23	•	
24	•	
25	•	
26	•	
27	•	
28	•	

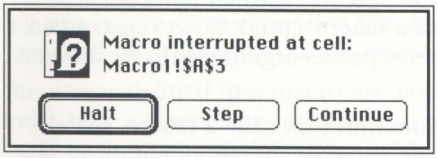
You can set up a WHILE-DO loop in the following manner:

Macro1

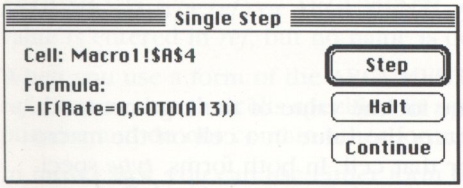
	A	B
11	•	
12	•	
13	while	
14	=IF(NOT(N=0),GOTO(continue))	
15	•	
16	•	
17	=GOTO(while)	
18	continue	
19	•	
20	•	
21	•	
22	•	
23	•	
24	•	
25	•	
26	•	
27	•	
28	•	

Interrupting a Macro

While a macro is running, you can interrupt it by pressing Command-.(period). When you do so, Excel displays a dialog box that tells you which cell in the macro Excel was about to calculate and and asks if you want to halt the macro, continue with the macro, or step through the macro.



The dialog box contains Halt, Step, and Continue buttons. If you click Halt, Excel halts the macro. If you click Continue, Excel continues normal calculation. If you click Step, Excel displays a dialog box that tells you which cell in the macro Excel is about to calculate, and the formula in that cell.



The single-step dialog box allows you to calculate a macro one cell at a time. This is a useful debugging tool.

If you click Step, Excel calculates the next cell in the macro and stops again. The dialog box stays on the screen so that you can click Step again. If you click Halt, Excel automatically halts the macro. If you click Continue, Excel continues normal calculation.

For a complete description of the single-step dialog box, see the STEP function in Chapter 8, "Macro Language Function Directory."

Passing Arguments and Returning Results

You can pass arguments to any macro that is called by a function in a worksheet or a macro sheet formula. For each argument to be passed to a macro, there must be one ARGUMENT function in the macro itself.

You cannot pass arguments to, or return results from, a macro that you run with the Run command (or with an Option-Command-key). Note that such a macro could, however, contain a function that calls another macro, passes arguments to it, and gets results back.

If you omit an argument in a function that calls a macro, Excel uses the error value #N/A as the value of the argument.

The ARGUMENT function must come before all other formulas in the macro, except the RESULT function, which, if used, must come first. (You may, however, have constants such as labels or comments before the ARGUMENT and RESULT functions.)

As described in Chapter 8, "Macro Language Function Directory," there are two forms for the ARGUMENT function:

ARGUMENT(*name,type*)
ARGUMENT(*name,type,ref*)

The first form defines a name for the value of the argument passed. The second form enters the value in a cell on the macro sheet and defines a name for that cell. In both forms, *type* specifies a type for the value to be passed. *Type* is optional. For more information, see "Argument Types" below.

The following three sections explain more about the forms of the ARGUMENT function, and the last two sections discuss results and types.

Defining Names for Arguments

With the form ARGUMENT(*name,type*), *name* is defined for the value of the argument passed. You can then use the name in other formulas in the macro.

The names defined by this form of the ARGUMENT function are just normal named values and references. They can be used in any formulas on the macro sheet. You can use the same names for different arguments in different macros on a macro sheet, but remember that a name can only have one definition at a time.

Therefore, if one macro calls another macro on the same macro sheet, the arguments to the two macros should have different names. Otherwise, when the second function is called, it will change the definitions of the names which may be used later in the first macro.

Be careful not to use names for arguments if those names are used elsewhere in the macro as non-argument names. For example, if a cell is named Months on your macro sheet because it is the first cell of a macro or because it is the destination of a GOTO function, then no arguments on the macro sheet should use that name.

The names defined in a macro are not deleted when the macro ends. After a macro has run, you can use the Define Name command to find out the definitions of the names used in the macro.

Entering Arguments in Cells

With the form ARGUMENT(*name,type,ref*), the value of the argument passed is entered as a constant in the cell specified by *ref*. Then, *name* is defined on the macro sheet for the reference. You can then use the name or the reference in other formulas in the macro. With this form, you can omit *name* and use the form ARGUMENT(,*type,ref*) or ARGUMENT(,,*ref*), in which case the value is entered in *ref*, but no name is defined for the reference.

When you use a form of the ARGUMENT function that includes *ref*, be sure that the cells specified in *ref* are available for that purpose and do not contain other formulas or constants.

Using ARGUMENT correctly

In most cases, you can use either form of ARGUMENT: either define a name for the value of the argument, or enter the value in a cell and, optionally, define a name for the cell.

If an argument is supposed to be a reference, you cannot use a form of the ARGUMENT function with *ref*. While a reference can have a name defined for it, it cannot be placed in a cell.

If the argument to be passed is an array, it is a good idea to define a name for the array, rather than entering it in cells on the macro sheet. That way, you do not need to reserve space on the macro sheet for the array. To obtain the values of individual elements in the array, use the INDEX function and the name of the array.

Occasionally, you may want to change individual values within an array argument in the course of your macro. You cannot do this if the array is just named. Instead, you must enter the array of values on the macro sheet, using a form of ARGUMENT with *ref*. Then you can use the SET.VALUE function to change individual values.

When you use *ref* in an ARGUMENT function, make sure the reference is the right size for the argument to be passed. If the argument is supposed to be a single value, then *ref* should be a reference to a single cell. If the argument is supposed to be an array, then *ref* must be a reference to a range of cells large enough to hold the entire array. You must ensure that there is enough room on the macro sheet for the largest array that might be passed as an argument. The size of *ref* in the ARGUMENT function will be the limit on the size of the array that can be passed as an argument. Make sure that anyone using the function is aware of the limit.

Even when an argument is supposed to be an array, you might not know in advance the size of the array that will be passed to the macro. You cannot pass an array that is larger than *ref*. But you can pass an array that is smaller than *ref*, in which case only some of the cells in *ref* are filled with values. When you use the form ARGUMENT(*name,type,ref*), *name* is defined for the reference that is actually filled by the array, not for all of *ref*. You can then use the ROWS and COLUMNS functions with the name to find out the actual size of the array passed.

Argument Types

When you include *type* in the ARGUMENT function, you are specifying that the argument passed should be of a certain type in order for the macro to run. If it is not, Excel first attempts to convert it to the specified type. If it cannot be converted, the macro returns the error value #VALUE!, without running.

The argument type is specified by a number as follows:

1	Number
2	Text
4	Logical
8	Reference
16	Error
64	Array

If you specify a type that is a sum of different numbers above, then the value passed can be of different types. For example, if you want to allow either a number or text to be passed as the value of an argument, then the type for that argument should be 1 + 2, or 3. Once a value for such an argument has been passed to the macro, you can find out its type with the worksheet function TYPE.

In all forms of the ARGUMENT function, you may omit *type*, in which case it is assumed to be 7; that is, single numbers, text, and logical values are allowed.

If an argument is supposed to be an array, its type should be 64. You should not specify a type such as 66; that is, a combination of the array type and one of the other types. Similarly, if an argument is supposed to be a reference, its type should be 8 and not a combination of that type and another type.

Result Types

In addition to specifying argument types, you can specify types for the values that a function returns. To do so, use the RESULT function. To return an array or a reference, you must use the RESULT function.

The RESULT function, if used, must come before any other formulas in the macro, including the ARGUMENT functions. (You may, however, have constants such as labels or comments before the ARGUMENT and RESULT functions.) The value that a macro returns, as specified in its RETURN function, should be of the type specified by the RESULT function. If it is not, Excel first attempts to convert it to the specified type. If it cannot be converted, the macro returns the error value #VALUE!.

The form of the RESULT function is

RESULT(*type*)

Type specifies the type of the return value as described above for the ARGUMENT function.

1	Number
2	Text
4	Logical
8	Reference
16	Error
64	Array

As with the *type* argument in the ARGUMENT function, you can omit the RESULT function, in which case the type of the result is assumed to be 7; that is, single numbers, text, and logical values are allowed. To return an array or a reference, you must use 64 or 8.

More on Types

Even though you have specified a type for an argument, the argument to the calling function does not have to be a constant of that type.

For example, an argument that is supposed to be a number could be supplied as a formula that evaluates to a number or a reference that contains a number. Perhaps a macro named SQUARE has one argument, specified to be a number. You can call it with the function SQUARE(2) or SQUARE(2+SUM(3,4,5)) or SQUARE(A1) if A1 contains a number. In the last case, because the argument is specified as the number type, Excel converts the reference A1 to the value stored at A1 and passes the value. The same is true for the other argument types. In particular, if an argument is specified as the array type, you can supply a reference to a range of cells as the argument. Excel will pass the array of values stored in that range.

Usually, when you supply a reference as an argument to a function, you actually want to pass the value(s) contained in that reference. Similarly, when you use a reference in the RETURN function, you usually want to return the value(s) contained in that reference. You will rarely specify the reference type for an argument or a result. However, there are times when you want to pass or return the reference itself. For example, the built-in worksheet function INDEX and the Macro Language function OFFSET accept references as arguments and give references as results.

When you pass a reference as an argument, it is passed as an external reference. For example, suppose you have a macro named Point on a macro sheet named Macros, and Point takes an argument that is specified to be a reference. If you enter the formula =Macros!Point(A1) on a worksheet named Budget, the reference that is passed to the Point function is the full external reference, Budget!A1.

Array type must be used to pass arrays

As discussed in Chapter 1, "Using Arrays," Excel allows you to give an array as an argument to a built-in function that otherwise works on single values. Excel simply assumes that you want the function to operate on all the elements in the array, instead of on a single value.

However, Excel does not make the same assumption when you are passing arguments to a function macro. That is, if an argument to a function macro is specified as a single-value type, the argument passed must be a single value. If you give an array for that argument, Excel will return the error value #VALUE!. Therefore, before you can pass an array argument to a function macro, you must specify the array type for that argument with the ARGUMENT function.

8 Macro Language Function Directory

This chapter is divided into three main sections: a brief discussion of the arguments expected by Excel's Macro Language functions, a concise listing of the available functions, divided by menu and subject categories, and a detailed alphabetical directory of these functions.

Arguments to Macro Language Functions

The functions in the Excel Macro Language use the same rules for syntax and argument definition as the worksheet functions.

In the Macro Language function directory, each argument is described by one of the following:

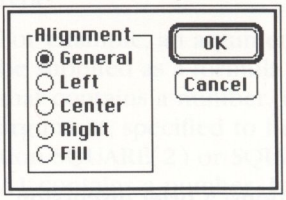
- Its type (i.e., number, numbers, text, logical, logicals, reference, value, values, array, or vector)
- A descriptive term, such as "window" or "outline," that is explained in the text
- A combination of the type and the descriptive term, such as "find-text" or "y-number"

The types of arguments are the same as those described for Excel's built-in functions (see Chapter 3, "Worksheet Function Directory").

Arguments to Command-Equivalent Functions

Most of the action-taking functions in the Excel Macro Language carry out an action that is equivalent to a command from the menu bar.

For example, ALIGNMENT(*type*) is the equivalent of the Alignment command from the Format menu. The argument, *type*, is the equivalent of the selection you make in the Alignment dialog box:



The argument is 1 for the first button (General), 2 for the second button (Left), 3 for the third button (Center), 4 for the fourth button (Right), and 5 for the fifth button (Fill).

In this directory, most of the arguments to command-equivalent functions have descriptive names like *type* to indicate the corresponding options from the dialog box.

Most command-equivalent functions also have an alternative, "question-mark" form, such as ALIGNMENT?( ), as shown in the Macro Language function directory. This form brings up a dialog box while the macro is running and pauses while you make the selection(s) you want and click OK. For more information, see "Making Command Macros Interactive" in Chapter 5, "Command Macros."

Macro Language Functions by Category

The functions in this directory are presented in alphabetical order for quick reference. However, to help you see their relationships with menu commands and regular Excel actions, they are also presented below, grouped by subject and menu categories.

Command-Equivalent Functions by Menu

File menu

File Menu

NEW(*type*)

OPEN(*document,update*)

OPEN.LINKS(*documents-1,documents-2,...*)

SAVE( )
SAVE.AS(*document,type*)
FILE.DELETE(*document*)
PAGE.SETUP(*header,footer,left,width,top,length,headings,message*)
PAGE.SETUP(*header,footer,left,right,top,bottom,headings,gridlines*)
PAGE.SETUP(*header,footer,left,width,top,height,size*)
PRINT(*range,from,to,copies,feed*)
PRINT( )
PRINTER.SETUP(*printer,port,baud*)

Edit Menu

Edit menu

UNDO( )
CUT( )
COPY( )
COPY.CHART(*as-shown*)
COPY.PICTURE( )
PASTE( )
CLEAR(*parts*)
PASTE.SPECIAL(*parts,operation*)
PASTE.SPECIAL(*values-in,series-names,categories*)
PASTE.SPECIAL(*parts*)
EDIT.DELETE(*direction*)
INSERT(*direction*)
FILL.RIGHT( )
FILL.DOWN( )

Formula Menu

Formula menu

DEFINE.NAME(*name,refers-to,type,key*)
DELETE.NAME(*name*)
CREATE.NAMES(*top-row,left-column*)
FORMULA.GOTO(*ref*)
FORMULA.FIND(*find-text,look-in,look-at,look-by*)
SELECT.LAST.CELL( )
SHOW.ACTIVE.CELL( )

Format menu

Format Menu

FORMAT.NUMBER(*format*)
DELETE.FORMAT(*format*)
ALIGNMENT(*type*)
STYLE(*bold,italic*)
BORDER(*outline,left,right,top,bottom*)
CELL.PROTECTION(*locked,hidden*)
COLUMN.WIDTH(*width*)

Data menu

Data Menu

DATA.FIND(*logical*)
EXTRACT(*unique*)
DATA.DELETE( )
SET.DATABASE( )
SET.CRITERIA( )
SORT(*sort-by,1st-key,order,2nd-key,order,3rd-key,order*)
DATA.SERIES(*series-in,type,unit,step,stop*)
TABLE(*row-input,column-input*)

Options menu

Options Menu

SET.PRINT.AREA( )
SET.PRINT.TITLES( )
SET.PAGE.BREAK( )
REMOVE.PAGE.BREAK( )
FONT(*name,size*)
DISPLAY(*formulas,gridlines,headings*)
PROTECT.DOCUMENT(*logical*)
PRECISION(*logical*)
A1.R1C1(*logical*)
CALCULATE.NOW( )
CALCULATION(*type,iteration,number-of,change*)

Window menu

Window Menu

SHOW.CLIPBOARD( )
NEW.WINDOW( )

Gallery Menu

Gallery menu

GALLERY.AREA(*type*)
GALLERY.BAR(*type*)
GALLERY.COLUMN(*type*)
GALLERY.LINE(*type*)
GALLERY.PIE(*type*)
GALLERY.SCATTER(*type*)
COMBINATION(*type*)
PREFERRED( )

Chart Menu

Chart menu

MAIN.CHART.TYPE(*type*)
OVERLAY.CHART.TYPE(*type*)
LEGEND(*logical*)
SELECT.CHART( )

Macro Menu

Macro menu

RUN(*ref*)

Action-Equivalent Functions

Action-equivalent functions

ACTIVATE(*window*)
Activates a window

ACTIVATE.NEXT( )
ACTIVATE.PREV( )
Activates the next and previous window

CANCEL.COPY( )
Cancels the Edit menu Copy command

CLOSE( )
Closes the active window

COLUMN.WIDTH(*width,ref*)
Changes the width of a column

DATA.FIND.NEXT( )
DATA.FIND.PREV( )
Finds the next and previous matching record in the database

FORMULA(*formula-text*)
Enters a formula in the selection

FORMULA.ARRAY(*formula-text*)
Enters an array formula

FORMULA.FILL(*formula-text*)
Enters a formula in the entire selection

FORMULA.FIND.NEXT( )
FORMULA.FIND.PREV( )
Finds the next and previous cell on the worksheet

FULL(*logical*)
Changes the size of the active window

HLINE(*number*)
VLINE(*number*)
Scrolls the active window by rows or columns

HPAGE(*number*)
VPAGE(*number*)
Scrolls the active window by window-fulls

HSCROLL(*column-number*)
VSCROLL(*row-number*)
Scrolls the active window by dragging the scroll box

MOVE(*x-number,y-number,window*)
Moves a window by dragging its title bar

SELECT(*selection-ref,active-cell-ref*)
Selects cells or changes the active cell

SIZE(*x-number,y-number,window*)
Changes the size of a window by dragging its size box

UNLOCKED.NEXT( )
UNLOCKED.PREV( )
Moves the active cell to next or previous unlocked cell in a protected worksheet

Control functions

Functions that Control Macro Calculation, Arguments, and User Interaction

ref(*argument-1,argument-2,...*)
Calls the macro at *ref*

ALERT(*text,type*)
Displays an alert box

ARGUMENT(*name,type*)
ARGUMENT(*name,type,ref*)
Passes an argument to a macro

BEEP( )
Sounds the Macintosh's bell

ECHO(*logical*)
Turns screen updating on and off

ERROR(*logical,ref*)
Turns error checking on and off

GOTO(*ref*)
Branches calculation to *ref*

HALT( )
Stops calculation

INPUT(*prompt,type,title*)
Displays a prompt and pauses for input

MESSAGE(*logical,text*)
Turns display of messages on and off

RESULT(*type*)
Specifies type of value returned by a function macro

RETURN(*values*)
RETURN( )
Ends the currently running macro and can return a value

SET.NAME(*name,value*)
Defines a name on the macro sheet

SET.VALUE(*ref,values*)
Changes the value(s) of cell(s) on the macro sheet

STEP( )
Turns single-stepping on

Special-Purpose Functions

Special-purpose functions

ABSREF(*ref-text,ref*)
Gives the reference that has *ref-text* relationship to *ref*

ACTIVE.CELL( )
Gives the reference of the active cell

CALLER( )
Gives the reference of the calling function

DEREF(*ref*)
Gives the value(s) of the cell(s) in *ref*

DOCUMENTS( )
Gives the names of all open documents

GET.FORMULA(*ref*)
Gives the contents of the formula bar for a cell

GET.NAME(*name*)
Gives the definition of the specified name

LINKS(*document*)
Gives the names of all worksheets referred to in *document*

OFFSET(*ref,row-offset,column-offset*)
Gives the offset reference with the same size as *ref*

RELREF(*ref-1,ref-2*)
Gives the relationship of *ref-1* to *ref-2*

SELECTION( )
Gives the reference of the selection

WINDOWS( )
Gives the names of all windows on the screen

ref

ref(argument-1,argument-2,...)
ref( )

When a reference followed by a set of parentheses is encountered in the calculation of a macro, calculation branches to the upper-left corner cell of that reference. The macro starting at that cell is said to be "called" by the macro containing the reference.

Ref can be a reference or a named reference. One macro can call another macro that is on another macro sheet using an external reference such as Finance!A1 or Finance!Future. Otherwise, ref is assumed to be on the same macro sheet as the calling macro.

The calling macro can pass from 0 to 13 arguments to the called macro.

A1.R1C1

A1.R1C1(logical)

The A1.R1C1 function is the equivalent of the A1 and R1C1 commands from the Options menu.

If *logical* is TRUE, Excel carries out the A1 command. If *logical* is FALSE, Excel carries out the R1C1 command.

ABSREF

ABSREF(ref-text,ref)

The ABSREF function gives the reference of the cells that have the relative relationship to *ref* that is specified by *ref-text*. *Ref-text* must be an R1C1-style relative reference in the form of text, such as "R[1]C[1]". *Ref-text* is considered relative to the upper-left corner cell of *ref*. *Ref* can be an external reference.

Examples

ABSREF("R[−2]C[−2]",C3) *equals* A1

ABSREF("R[−2]C[−2]",Finance!C3) *equals* Finance!A1

ABSREF("R[−2]C[−2]:R[2]C[2]",C3:G7)
equals ABSREF("R[−2]C[−2]:R[2]C[2]",C3)
equals A1:E5

ABSREF(RELREF(A1,C3),D4) *equals* B2

ACTIVATE(window)

ACTIVATE

The ACTIVATE function is equivalent to the action of activating a window.

Window should be the name of a window in the form of text, such as "Sales", or "Sales:1" if there is more than one window displaying the document named Sales.

If a document is displayed in more than one window, and if *window* is the name of the document rather than of one of the windows containing that document, the first window containing that document is activated.

ACTIVATE.NEXT( )
ACTIVATE.PREV( )

ACTIVATE.NEXT
ACTIVATE.PREV

The ACTIVATE.NEXT and ACTIVATE.PREV functions are equivalent to the actions of pressing Command-M and Command-Shift-M. They activate the next and previous windows.

ACTIVE.CELL( )

ACTIVE.CELL

The ACTIVE.CELL function gives the reference of the active cell in the selection as an external reference.

See also SELECTION.

Example

If the document in the active window is named Sales, and if A1 is the active cell, then:

ACTIVE.CELL( ) *equals* Sales!A1

ALERT

ALERT(text,type)

The ALERT function displays an alert box and waits for you to click a button. The alert box contains the message specified in *text*, and an icon and buttons determined by *type*.

The types are:

1 "Caution" alert with a "?" icon and OK and Cancel buttons

2 "Note" alert with an "*" icon and an OK button

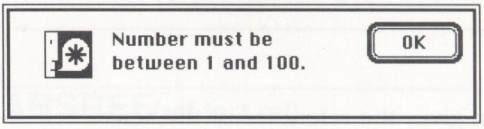
3 "Stop" alert with an "!" icon and an OK button

If you click OK, ALERT gives the logical value TRUE. If you click Cancel, ALERT gives the logical value FALSE.

Examples

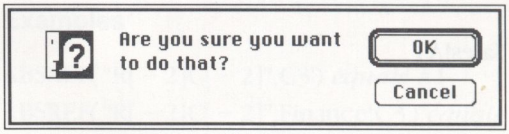
The following formula displays the alert box shown below:

ALERT("Number must be between 1 and 100.",2)



The following formula displays the alert box shown below:

ALERT("Are you sure you want to do that?",1)



ALIGNMENT(type)
ALIGNMENT?( )

ALIGNMENT

The ALIGNMENT function is the equivalent of the Alignment command from the Format menu.

Type must be a number:

1 General
2 Left
3 Center
4 Right
5 Fill

ARGUMENT(name,type)
ARGUMENT(name,type,ref)

ARGUMENT

The ARGUMENT function allows arguments to be passed to a macro that is called from a function on a worksheet or from another macro.

For each argument to your macro, there must be one ARGUMENT function in the macro itself. If a macro contains an ARGUMENT function and you omit the corresponding argument in the function calling the macro, Excel uses the error value #N/A as the value of the argument. The ARGUMENT functions must come before any other formulas in the macro, except the RESULT function.

If you use the form ARGUMENT(*name,type*), the specified name is defined on the macro sheet for the value passed. Then, you can use *name* in formulas in the macro.

If you use the form ARGUMENT(*name,type,ref*), the value passed is entered as a constant in the cell specified by *ref*. The specified name is defined on the macro sheet for *ref*. Then, you can refer to that cell in the formulas with either *ref* or *name*. You can also omit *name* and use the form ARGUMENT(,*type,ref*).

If an argument is supposed to be a reference, then the form ARGUMENT(*name,type*) must be used, rather than ARGUMENT(*name,type,ref*), because a reference cannot be placed in a cell.

Type determines what values Excel will accept for the argument. The value passed should be of the specified type. If it is not, Excel first attempts to convert it to the specified type. If the value cannot be converted, the macro returns the error value #VALUE! without being calculated.

Type must be a number:

1	Number
2	Text
4	Logical
8	Reference
16	Error
64	Array

Type can be a sum of the different numbers above, in which case the value passed can be of different types. For example, if *type* is 7, which is the sum of 1, 2, and 4, then the value passed can be a single number, text, or logical value.

If you omit *type*, it is assumed to be 7. If an argument is supposed to be a reference, *type* should be 8. If an argument is supposed to be an array, *type* should be 64.

For more information on using the ARGUMENT function, see "Passing Arguments and Returning Results" in Chapter 7, "More Information about Macro Sheets."

BEEP

BEEP( )

The BEEP function causes Excel to sound the Macintosh's bell. Typical uses of BEEP are to signal the end of a macro and to alert the user to a new INPUT dialog box caused by the INPUT function. BEEP produces the same sound produced by alert boxes and by clicking outside a dialog box. The volume can be controlled with the Control Panel desk accessory.

BORDER(outline,left,right,top,bottom)
BORDER?( )

BORDER

The BORDER function is the equivalent of the Border command from the Format menu.

Outline, *left*, *right*, *top*, and *bottom* must be logical values:

TRUE	Checked
FALSE	Not checked

CALCULATE.NOW( )

CALCULATE.NOW

The CALCULATE.NOW function is the equivalent of the Calculate Now command from the Options menu.

CALCULATION(type,iteration,number-of,change)
CALCULATION?( )

CALCULATION

The CALCULATION function is the equivalent of the Calculation command from the Options menu for worksheets and from the Chart menu for charts.

Type must be a number:

1	Automatic
2	Automatic Except Tables
3	Manual

Iteration must be a logical value:

TRUE	Checked
FALSE	Not checked

Number-of and *change* must be numbers.

CALLER

CALLER( )

The CALLER function gives the reference of the cell containing the function that called the currently running macro. If the function was part of an array formula entered into a range of cells, CALLER gives the reference of the range. If the macro was called with the Run command (or with an Option-Command-key), CALLER gives the error value #REF!.

CALLER is useful in a macro whose behavior depends on the location or size of the calling reference.

Examples

In a macro that is called from a function in an array formula, the following functions give the number of rows and columns, respectively, in the calling reference:

```
ROWS(CALLER( ))
COLUMNS(CALLER( ))
```

CELL.PROTECTION

CELL.PROTECTION(locked,hidden)
CELL.PROTECTION?( )

The CELL.PROTECTION function is the equivalent of the Cell Protection command from the Format menu.

Locked and *hidden* must be logical values:

TRUE	Checked
FALSE	Not checked

CLEAR

CLEAR(parts)
CLEAR?( )

The CLEAR function is the equivalent of the Clear command from the Edit menu.

Parts must be a number:

1	All
2	Formats
3	Formulas

CLOSE( )

CLOSE

The CLOSE function is equivalent to the action of clicking the close box of the active window. It closes the active window.

COLUMN.WIDTH(width)
COLUMN.WIDTH(width,ref)
COLUMN.WIDTH?( )

COLUMN.WIDTH

The COLUMN.WIDTH function is equivalent to the action of changing the width of a column. The COLUMN.WIDTH(*width*) form is the equivalent of the Column Width command from the Format menu, which changes the width of the column(s) containing the selection. The COLUMN.WIDTH(*width,ref*) form is equivalent to the action of dragging the border of the specified column on the active worksheet.

Width must be a number.

If *ref* is specified, it should be either an external reference to the active worksheet, such as !$A:$C or !Database, or an R1C1-style reference in the form of text, such as "C1:C3", "C[−4]:C[−2]", or "Database". If *ref* is a relative reference in the form of text, it is assumed to be relative to the active cell in the selection.

If you are recording a macro, and you change column widths by dragging, Excel records the reference of the columns using R1C1-style references in the form of text. If you are recording with Relative Record on, Excel uses R1C1-style relative references. If you are recording with Absolute Record on, Excel uses R1C1-style absolute references.

Examples

If the selection is the range C1:E50, this function changes the width of columns C:E to 15:

```
COLUMN.WIDTH(15)
```

If the active cell is C1, this function makes columns D:F zero-width columns:

```
COLUMN.WIDTH(0,"C[1]:C[3]")
```

Each of the following functions changes the width of columns C and D to 15:

```
COLUMN.WIDTH(15,!$C:$D)
COLUMN.WIDTH(15,"C3:D4")
```

COMBINATION

COMBINATION(type)
COMBINATION?( )

The COMBINATION function is the equivalent of the Combination command from the Gallery menu.

Type must be a number corresponding to a format in the gallery.

COPY
CANCEL.COPY

COPY( )
CANCEL.COPY( )

The COPY function is the equivalent of the Copy command from the Edit menu. The CANCEL.COPY function is equivalent to the action of pressing Command-.(period) to cancel the marquee after you copy a selection.

COPY.CHART

COPY.CHART(as-shown)
COPY.CHART?( )

The COPY.CHART function is the equivalent of the Copy Chart command from the Edit menu.

As-shown must be a number:

1 As Shown on Screen

2 As Shown when Printed

COPY.PICTURE

COPY.PICTURE( )

The COPY.PICTURE command is equivalent to the action of choosing the Copy command from the Edit menu while pressing the Shift key. It copies the selection to the Clipboard as a picture.

CREATE.NAMES(top-row,left-column)
CREATE.NAMES?( )

CREATE.NAMES

The CREATE.NAMES function is the equivalent of the Create Names command from the Formula menu.

Top-row and *left-column* must be logical values:

TRUE	Checked
FALSE	Not checked

CUT( )

CUT

The CUT function is the equivalent of the Cut command from the Edit menu.

DATA.DELETE( )
DATA.DELETE?( )

DATA.DELETE

The DATA.DELETE function is the equivalent of the Delete command from the Data menu.

DATA.FIND(logical)

DATA.FIND

The DATA.FIND function is the equivalent of the Find and Exit Find commands from the Data menu.

If *logical* is TRUE, Excel carries out the Find command. If *logical* is FALSE, Excel carries out the Exit Find command.

DATA.FIND.NEXT( )
DATA.FIND.PREV( )

DATA.FIND.NEXT
DATA.FIND.PREV

The DATA.FIND.NEXT and DATA.FIND.PREV functions are equivalent to the actions of pressing Command-F and Command-Shift-F. They find the next and previous matching records in the database. If the function cannot find a matching record, it gives the logical value FALSE.

DATA.SERIES

DATA.SERIES(series-in,type,unit,step,stop)
DATA.SERIES?( )

The DATA.SERIES function is the equivalent of the Series command from the Data menu.

Series-in must be a number:

1	Rows
2	Columns

Type must be a number:

1	Linear
2	Growth
3	Date

Unit must be a number:

1	Day
2	Weekday
3	Month
4	Year

Step and *stop* must be numbers.

DEFINE.NAME

DEFINE.NAME(name,refers-to,type,key)
DEFINE.NAME?( )

The DEFINE.NAME function is the equivalent of the Define Name command from the Formula menu. It defines the specified name on the active worksheet.

Name must be text.

With the Define Name command, you can define names for references, values, and formulas. Accordingly, *refers-to* can be a reference, number, text, or logical value, or a formula in the form of text, such as "=2*PI( )/360".

If *refers-to* is a normal external reference to cells, such as !$A$1 or Sales!$A$1:$C$3, Excel defines *name* to refer to those cells.

If you omit *refers-to*, Excel defines *name* to refer to the cells in the selection.

If *refers-to* is a number, text, or logical value, Excel defines *name* to refer to that value.

If *refers-to* is a formula in the form of text that contains references, those references must be R1C1-style references, such as "=R2C2*(1+RC[-1])". If you are recording a macro, and you define a name to refer to a formula, Excel converts A1-style references to R1C1-style references. For example, if the active cell is C2, and you define the name Previous to refer to =B2, Excel records that command as DEFINE.NAME("Previous","=RC[-1]").

The last two arguments apply only if the document in the active window is a macro sheet.

Type must be a number:

1	Function
2	Command

Key must be a single letter in the form of text, such as "Z".

Examples

This function defines the name Inflation on the active worksheet to refer to 0.12:

```
DEFINE.NAME("Inflation",0.12)
```

If the selection is A1:C3, each of the following functions defines the name Sales on the active worksheet to refer to cells A1:C3:

```
DEFINE.NAME("Sales")
DEFINE.NAME("Sales",SELECTION( ))
```

This function defines the name Previous to refer to the relative reference RC[-1]:

```
DEFINE.NAME("Previous","=RC[-1]")
```

This function defines the name Sales on the active worksheet to refer to A1:C1, the first row of A1:C3:

```
DEFINE.NAME("Sales1",INDEX(SELECTION( ),1,0))
```

DELETE.FORMAT

DELETE.FORMAT(format)

The DELETE.FORMAT function is equivalent to the action of deleting a format with the Number command from the Format menu. It deletes the specified format.

Format must be text, for example, "#,##0.00".

DELETE.NAME

DELETE.NAME(name)

The DELETE.NAME function is equivalent to the action of deleting a name with the Define Name command from the Formula menu. It deletes the specified name.

Name must be text.

DEREF

DEREF(ref)

The DEREF function gives the value of the cells in *ref*. If *ref* is the reference of a single cell, DEREF gives the value of that cell. If *ref* is the reference of a range of cells, DEREF gives the array of values in those cells.

In most formulas, there is no difference between using a value and using the reference of a cell which has that value. The reference is automatically converted to the value, as necessary. For example, if cell A1 contains the value 2, the formula =A1+1, like the formula =2+1, gives the result 3, because the reference A1 is converted to the value 2. However, in a few functions, such as the SET.NAME function, references are not automatically converted to values. Instead, those functions behave differently depending on whether an argument is a reference or a value.

For an example of the use of this function, see SET.NAME.

DISPLAY

DISPLAY(formulas,gridlines,headings)
DISPLAY?( )

The DISPLAY function is the equivalent of the Display command from the Options menu.

Formulas, *gridlines*, and *headings* must be logical values:

TRUE	Checked
FALSE	Not checked

DOCUMENTS( )

DOCUMENTS

The DOCUMENTS function gives, as an array of text values, the names of all the open documents, in alphabetical order. With the INDEX function, you can select individual document names from the array for use in other functions that take document names as arguments.

Example

If your screen contains the windows named Budget, Chart1, Actual:1, and Actual:2, which contain the documents named Budget, Chart1, and Actual, then:

DOCUMENTS( ) *equals* {"Budget","Chart1","Actual"}

ECHO(logical)

ECHO

The ECHO function controls screen updating while a macro is calculating. If *logical* is FALSE, Excel turns off screen updating. If *logical* is TRUE, Excel turns updating on and updates the screen.

When a macro ends, for whatever reason, Excel always updates the screen. Command macros run faster when screen updating is turned off; however, you cannot see what the macro is doing.

EDIT.DELETE(direction)
EDIT.DELETE?( )

EDIT.DELETE

The EDIT.DELETE function is the equivalent of the Delete command from the Edit menu.

Direction must be a number:

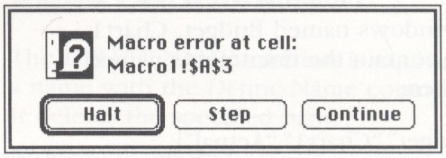
1 Shift Cells Left

2 Shift Cells Up

ERROR

ERROR(logical,ref)

The ERROR function turns error checking on and off. If *logical* is TRUE, Excel turns on error checking. Then, if an error is encountered in the calculation of the macro, calculation branches to the upper-left corner cell of *ref*. If you omit *ref*, Excel instead displays a dialog box that tells you in which cell the error has occurred and asks you if you want macro calculation to continue.



The dialog box contains Halt, Step, and Continue buttons. If you click Halt, Excel automatically halts the macro. If you click Step, Excel begins single-step calculations. If you click Continue, Excel continues normal calculation.

If *logical* is FALSE, Excel turns off error checking. Then, if an error is encountered in the calculation of the macro, Excel ignores it, and calculation continues.

EXTRACT

EXTRACT(unique)
EXTRACT?( )

The EXTRACT function is the equivalent of the Extract command from the Data menu.

Unique must be a logical value:

TRUE	Checked
FALSE	Not checked

FILE.DELETE(document)
FILE.DELETE?( )

FILE.DELETE

The FILE.DELETE function is the equivalent of the Delete command from the File menu.

Document must be the name of a document in the form of text, such as "Sales".

If *document* is not on the disk in the default disk drive, Excel displays a dialog box requesting that you insert a disk containing the specified document.

FILL.DOWN( )
FILL.RIGHT( )

FILL.DOWN
FILL.RIGHT

The FILL.DOWN and FILL.RIGHT functions are the equivalents of the Fill Down and Fill Right commands from the Edit menu.

FONT(name,size)
FONT?( )

FONT

The FONT function is the equivalent of the Font command from the Options menu.

Name must be the name of a font in the form of text, such as "Geneva".

Size must be a number.

FORMAT.NUMBER(format)
FORMAT.NUMBER?( )

FORMAT.NUMBER

The FORMAT.NUMBER function is the equivalent of the Number command from the Format menu.

Format must be text, for example, "#,##0.00".

FORMULA

FORMULA(formula-text)

The FORMULA function is equivalent to the action of entering a formula in the active cell. It enters the specified formula in the active cell. The formula is entered just as if you typed it in the formula bar and pressed the Enter key.

Formula-text can be a formula in the form of text, such as "=2*PI( )/360", or a number, text, or logical value. If *formula-text* is a formula, that formula is entered. If *formula-text* is a number, text, or logical value, that value is entered as a constant.

If *formula-text* contains references, those references must be R1C1-style references, such as "=RC[-1]*(1+R1C1)". If you are recording a macro when you enter a formula, Excel converts A1-style references to R1C1-style references. For example, if you enter the formula =B2*(1+$A$1) in cell C2, Excel records that action as FORMULA("=RC[-1]*(1+R1C1)").

Examples

This function enters the constant number 523 in the active cell:

```
FORMULA(523)
```

If the active cell is C2, this function enters the formula =RC[-1]*(1+R1C1), or =B2*(1+$A$1), in cell C2:

```
FORMULA("=RC[-1]*(1+R1C1)")
```

This function takes the result of the INPUT function and enters it in the active cell:

```
FORMULA(INPUT("Enter a formula:",0))
```

FORMULA.ARRAY

FORMULA.ARRAY(formula-text)

The FORMULA.ARRAY function is equivalent to the action of entering an array formula while pressing the Command key. It enters the specified formula as an array formula in the selection.

For information on *formula-text*, see FORMULA.

FORMULA.FILL(formula-text)

FORMULA.FILL

The FORMULA.FILL function is equivalent to the action of entering a formula while pressing the Option key. It enters the specified formula in the entire selection.

For information on *formula-text*, see FORMULA.

FORMULA.FIND(find-text,look-in,look-at,look-by,direction)
FORMULA.FIND?( )

FORMULA.FIND

The FORMULA.FIND function is the equivalent of the Find command from the Formula menu. Except for *find-text*, each FORMULA.FIND argument must be either the number 1 or the number 2. The table below shows what each number finds for each argument. If a matching cell is not found, FORMULA.FIND gives the logical value FALSE.

Look-in:

1	Formulas
2	Values

Look-by:

1	Rows
2	Columns

Look-at:

1	Whole
2	Part

Direction:

1	Next
2	Previous

If you omit *direction*, it is assumed to be 1.

FORMULA.FIND.NEXT( )
FORMULA.FIND.PREV( )

FORMULA.FIND.NEXT
FORMULA.FIND.PREV

The FORMULA.FIND.NEXT and FORMULA.FIND.PREV functions are equivalent to the actions of pressing Command-H and Command-Shift-H. They find the next and previous cells on the worksheet, as specified in the Find dialog box from the Formula menu. If a matching cell is not found, the function gives the logical value FALSE.

FORMULA.GOTO

FORMULA.GOTO(ref)
FORMULA.GOTO?( )

The FORMULA.GOTO function is the equivalent of the Goto command from the Formula menu.

Ref should be either an external reference to a worksheet, such as !$A$1, !Price, or Sales!$A$1, or an R1C1-style reference in the form of text, such as "R1C1", "Price", or "Sales!R1C1".

If you are recording a macro, the reference you enter in the Goto command dialog box is recorded as text, with A1-style references converted to R1C1-style references.

Examples

Each of the following functions goes to cell A1 on the active worksheet:

```
FORMULA.GOTO(!$A$1)
FORMULA.GOTO("R1C1")
```

Each of the following functions goes to the cells named Sales on the active worksheet:

```
FORMULA.GOTO(!Sales)
FORMULA.GOTO("Sales")
```

Each of the following functions goes to cells B2:C3 on the worksheet named Budget:

```
FORMULA.GOTO(Budget!$B$2:$C$3)
FORMULA.GOTO("Budget!R2C2:R3C3")
```

FULL

FULL(logical)

The FULL function is equivalent to the action of changing the active window from full-size to its normal size or from its normal size to full-size by double-clicking its title bar.

If *logical* is TRUE, Excel makes the active window full-size. If *logical* is FALSE, Excel returns the active window to its previous normal size.

GALLERY.AREA(type)
GALLERY.AREA?( )

GALLERY.AREA

The GALLERY.AREA function is the equivalent of the Area command from the Gallery menu.

Type must be a number corresponding to a format in the gallery.

GALLERY.BAR(type)
GALLERY.BAR?( )

GALLERY.BAR

The GALLERY.BAR function is the equivalent of the Bar command from the Gallery menu.

Type must be a number corresponding to a format in the gallery.

GALLERY.COLUMN(type)
GALLERY.COLUMN?( )

GALLERY.COLUMN

The GALLERY.COLUMN function is the equivalent of the Column command from the Gallery menu.

Type must be a number corresponding to a format in the gallery.

GALLERY.LINE(type)
GALLERY.LINE?( )

GALLERY.LINE

The GALLERY.LINE function is the equivalent of the Line command from the Gallery menu.

Type must be a number corresponding to a format in the gallery.

GALLERY.PIE(type)
GALLERY.PIE?( )

GALLERY.PIE

The GALLERY.PIE function is the equivalent of the Pie command from the Gallery menu.

Type must be a number corresponding to a format in the gallery.

GALLERY.SCATTER

GALLERY.SCATTER(type)
GALLERY.SCATTER?( )

The GALLERY.SCATTER function is the equivalent of the Scatter command from the Gallery menu.

Type must be a number corresponding to a format in the gallery.

GET.FORMULA

GET.FORMULA(ref)

The GET.FORMULA function gives the contents of the upper-left corner cell in *ref* as they would appear in the formula bar. The contents are given in the form of text, for example, "=2*PI( )/360". *Ref* can be an external reference.

If the formula contains references, they are given as R1C1-style references, such as "=RC[−1]*(1+R1C1)".

Examples

If cell A3 on the active worksheet contains the number 523, then:

GET.FORMULA(!A3) *equals* "523"

If cell C2 on a worksheet named Budget contains the formula =B2*(1+$A$1), then:

GET.FORMULA(BUDGET!C2) *equals* "=RC[−1]*(1+R1C1)"

GET.NAME

GET.NAME(name)

The GET.NAME function gives the definition of the specified name as it would appear in the Refers To box in the Define Name command. The definition is given in the form of text.

Name must be text. *Name* can be a name defined on the macro sheet, an external reference to a name defined on the active worksheet, or an external reference to a name defined on a particular worksheet

If the name definition contains references, they are given as R1C1-style references, such as "RC[−1]*(1+R1C1)".

Examples

If the name Sales on a macro sheet is defined to be the constant number 523, then on that macro sheet:

GET.NAME("Sales") *equals* "=523"

If the name Profit on the active worksheet is defined to be the formula =Sales−Costs, then:

GET.NAME("!Profit") *equals* "=Sales−Costs"

If the name Database on a worksheet named Stocks is defined to be the range A1:F500, then:

GET.NAME("Stocks!Database") *equals* "=R1C1:R500C6"

GOTO(ref)

GOTO

The GOTO function causes calculation of a macro to branch to the upper-left corner cell of *ref*.

Ref can be an external reference to another macro sheet. If that macro sheet is not open, GOTO displays an alert box.

Example

On a macro sheet, if A1 contains the error value #N/A!, and if C3 contains the formula ALERT("Not available.", 2), then when the following formula is calculated, calculation branches to C3 and Excel displays an alert:

=IF(ISERROR(A1),GOTO(C3),GOTO(D4))

HALT( )

HALT

The HALT function stops calculation of the currently running macro. If the current macro is nested within other macros, those macros will halt as well.

Example

If A1 contains the error value #N/A!, then when the following formula is calculated, all macro calculation halts:

=IF(ISERROR(A1),HALT(C3)GOTO(D4))

HLINE
VLINE

HLINE(number)
VLINE(number)

The HLINE and VLINE functions are equivalent to the actions of scrolling the active window by columns or rows by clicking the scroll arrow. They scroll the active window horizontally and vertically by the specified number of columns or rows.

Number should be an integer. If *number* is positive, Excel scrolls to the right or down by that number of lines. If *number* is negative, Excel scrolls to the left or up by that number of lines.

HPAGE
VPAGE

HPAGE(number)
VPAGE(number)

The HPAGE and VPAGE functions are equivalent to the actions of scrolling the active window by window-fulls by clicking in the grey area of the scroll bar. They scroll the active window horizontally and vertically by the specified number of window-fulls.

Number should be an integer. If *number* is positive, Excel scrolls to the right or down by that number of window-fulls. If *number* is negative, Excel scrolls to the left or up by that number of window-fulls.

HSCROLL
VSCROLL

HSCROLL(column-number)
VSCROLL(row-number)

The HSCROLL and VSCROLL functions are equivalent to the actions of scrolling the active window by dragging the scroll box. They scroll the active window horizontally and vertically to the specified column or row.

Column-number and *row-number* should be percentages between 0 and 100 (or numbers between 0 and 1). If *column-number* is 0, HSCROLL scrolls to the first column, the left of the horizontal scroll bar. If *column-number* is 1, HSCROLL scrolls to the last column, the right of the scroll bar. Similarly, if *row-number* is 0 or 1, VSCROLL scrolls to row 1 or row 16384, respectively.

Examples

This function scrolls to column 128, or DX:

```
HSCROLL(128/256)
```

This function scrolls to row 8192, 50% of the way down the worksheet:

```
VSCROLL(50%)
```

INPUT(prompt,type,title)

INPUT

The INPUT function displays a dialog box into which you can input information and then gives that information as its result. INPUT provides a very powerful and flexible method for a macro to get information from the user.

The dialog box contains the prompt specified by *prompt*, the title specified by *title*, an edit box into which you enter your input, and OK and Cancel buttons. When the dialog box appears, you enter your input into the edit box.

The INPUT dialog box and edit box are very similar to those in the Define Name command from the Formula menu. You can move the INPUT dialog box by dragging its title bar. You can enter cell references in the edit box by selecting cells, and you can choose formula editing commands such as Paste Name and Paste Function.

If you click OK or press the Enter key, INPUT gives the value of what you have entered in the edit box. If you click Cancel or press Command-.(period), INPUT gives the logical value FALSE.

Prompt and *title* must be text. If you omit *title*, it is assumed to be the text Input.

Type determines exactly what the INPUT function gives. *Type* must be a number:

0	Formula
1	Number
2	Text
4	Logical
8	Reference
16	Error
64	Array

If *type* is 8, INPUT gives an absolute reference to the specified cells.

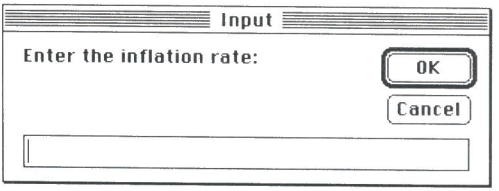
If *type* is 0, INPUT gives the contents of the edit box in the form of text, for example, "=2*PI( )/360". If the formula contains references, they are given as R1C1-style references, for example, "=RC[−1]*(1+R1C1)".

The input in the edit box should be of the specified type. If it is not, Excel first attempts to convert it to the specified type. If the input cannot be converted, Excel displays an alert box informing you that the input is of the wrong type.

Examples

The following function displays the dialog box shown below:

INPUT("Enter the inflation rate:",1)



If you then type 12%, INPUT gives the value 0.12.

If in response to the function INPUT("Select your database.",8) you select cells A1:C3 on the worksheet named Budget, INPUT gives the reference BUDGET!A1:C3.

If the active cell is C2, and in response to the function INPUT("Enter your monthly increase formula:",0) you enter the formula =B2*(1+$A$1), INPUT gives "=RC[-1]*(1+R1C1)".

INSERT(direction)
INSERT?( )

INSERT

The INSERT function is the equivalent of the Insert command from the Edit menu.

Direction must be a number:

1 Shift Cells Right

2 Shift Cells Down

LEGEND(logical)

LEGEND

The LEGEND function is the equivalent of the Add and Delete Legend commands from the Chart menu.

If *logical* is TRUE, Excel carries out the Add Legend command. If *logical* is FALSE, Excel carries out the Delete Legend command.

LINKS(document)

LINKS

The LINKS function gives, as an array of text values, the names of all worksheets referred to by external references in the specified document. With the INDEX function, you can select individual worksheet names from the array for use in other functions that take document names as arguments.

Document must be the name of a document in the form of text, such as "Sales". If *document* is omitted, it is assumed to be the name of the active document.

If the active document contains no external references, LINKS gives the error value #N/A.

Example

If the chart named Performance is open and contains data series that refer to worksheets named Budget and Actual, then when this function is calculated, Excel opens the worksheets named Budget and Actual:

```
OPEN.LINKS (LINKS("Performance"))
```

MAIN.CHART.TYPE

MAIN.CHART.TYPE(type)
MAIN.CHART.TYPE?( )

The MAIN.CHART.TYPE function is the equivalent of the Main Chart Type command from the Chart menu.

Type must be a number:

1	Area
2	Bar
3	Column
4	Line
5	Pie
6	Scatter

MESSAGE

MESSAGE(logical,text)

The MESSAGE function controls whether or not a message is displayed on the screen. MESSAGE is useful for informing the user about what the macro is doing.

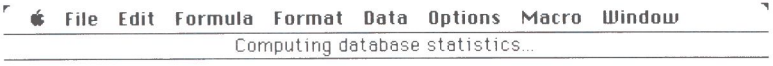
If *logical* is TRUE, Excel displays the specified text centered in a line above the formula bar. If *logical* is TRUE but *text* is omitted, Excel leaves the message line up, but displays no text in it. If *logical* is FALSE, Excel takes down the message.

In general, if a macro displays several different messages during the course of calculation, it should not take down the message line until after the macro is finished. If you want to display no message during some part of the macro, use the function MESSAGE(TRUE,"").

Example

The following function displays the message shown below:

MESSAGE(TRUE,"Computing database statistics...")



MOVE(x-number,y-number,window)

MOVE

The MOVE function is equivalent to the action of moving a window by dragging its title bar. It moves the specified window so that its upper-left corner is at the position specified by *x-number* and *y-number*. MOVE does not change the size of the window or affect whether the specified window is active or inactive.

X-number is the horizontal distance of the window from the left edge of the screen. The left edge of the screen has an *x-number* of 1, while the right edge has an *x-number* of 512.

Y-number is the vertical distance of the window from the top of the screen. The top of the screen has a *y-number* of 1, while the bottom has a *y-number* of 342. *Y-number* must be larger than 40, which is the *y-number* of the bottom of the formula bar.

Window must be the name of a window in the form of text, such as "Sales", or "Sales:1" if there is more than one window displaying the document named Sales. If *window* is "Clipboard", Excel moves the Clipboard. If you omit *window*, Excel moves the active window.

See also SIZE.

Examples

This function moves the active window so that its upper-left corner is at *x-number* 200, approximately two-fifths of the way across the screen, and *y-number* 110, approximately one-third of the way down the screen:

MOVE(200,110)

This function does the same thing to the document named Sales:

MOVE(200,110,"Sales")

NEW

NEW(type)
NEW?( )

The NEW function is the equivalent of the New command from the File menu.

Type must be a number:

1	Worksheet
2	Chart
3	Macro Sheet

NEW.WINDOW

NEW.WINDOW( )

The NEW.WINDOW function is the equivalent of the New Window command from the Window menu.

OFFSET

OFFSET(ref,row-offset,column-offset)

The OFFSET function gives a reference of the same size and shape as *ref*, with its upper-left corner shifted by *row-offset* and *column-offset*.

Row-offset and *column-offset* must be numbers. If *row-offset* is positive or negative, OFFSET gives a reference below or above *ref*. If *row-offset* is 0, OFFSET gives a reference that begins at the same row as *ref*.

Similarly, if *column-offset* is positive or negative, OFFSET gives a reference to the right or left of *ref*. If *column-offset* is 0, OFFSET gives a reference that begins at the same column as *ref*.

If *row-offset* and *column-offset* offset *ref* over the edge of the worksheet, OFFSET gives the error value #REF!.

Examples

OFFSET(C3,2,3) *equals* F5

OFFSET(C3:E5, −1,0) *equals* C2:E4

OFFSET(C3:E5,0, −3) *equals* #REF!

OPEN(document,update)
OPEN?( )

OPEN

The OPEN function is the equivalent of the Open command from the File menu.

Document must be the name of a document in the form of text, such as "Sales". If *document* is not on the disk in the default disk drive, Excel displays a dialog box so you can change disks.

Document can also contain the name of the disk where you want Excel to look for the document. For example, if *document* is "Data:Sales", Excel first looks for Sales on the disk named Data. If the Data disk is not in a disk drive, Excel displays a dialog box so you can change disks. This feature is especially useful if you have documents stored on different disks on a hard disk drive.

Excel uses *update* only if the specified document contains external references. If *update* is TRUE, Excel updates the external references; if *update* is FALSE, it does not. If *update* is omitted, Excel displays a dialog box asking whether or not to update the external references.

OPEN.LINKS(documents-1,documents-2,...)
OPEN.LINKS?( )

OPEN.LINKS

The OPEN.LINKS function is the equivalent of the Open Links command from the File menu.

OPEN.LINKS can have from 1 to 14 arguments. The arguments should be the names of documents in the form of text, or arrays or references that contain text, such as "Sales" or {"Budget","Actual"}.

If one of the specified documents contains external references to a supporting worksheet, OPEN.LINKS assumes that you want to update the external references. If you do not want to update them, use the OPEN function instead.

You can generate an array of the names of linked documents with the LINKS function.

Example

If a chart named Performance is open and contains data series that refer to worksheets named Budget and Actual, Excel opens Budget and Actual when this formula is calculated:

```
=OPEN.LINKS(LINKS( ))
```

OVERLAY.CHART.TYPE

OVERLAY.CHART.TYPE(type)
OVERLAY.CHART.TYPE?( )

The OVERLAY.CHART.TYPE function is the equivalent of the Overlay Chart Type command from the Chart menu.

Type must be a number:

1	None
2	Area
3	Bar
4	Column
5	Line
6	Pie
7	Scatter

PAGE.SETUP

PAGE.SETUP(header,footer,left,width,top,length, headings,message)
PAGE.SETUP(header,footer,left,right,top,bottom, headings,gridlines)
PAGE.SETUP(header,footer,left,width,top,height,size)
PAGE.SETUP?( )

The PAGE.SETUP function is the equivalent of the Page Setup command from the File menu. If the document in the active window is a worksheet, and if TTY is the printer specified in the Printer Setup command, use the first form of the function. If Macintosh is specified, use the second form. If the document in the active window is a chart, use the third form of the function.

Header and *footer* must be text.

Left, *right*, *top*, *bottom*, *width*, *length*, and *height* must be numbers.

Headings and *gridlines* must be logical values:

TRUE	Checked
FALSE	Not checked

Message must be text.

Size must be a number:

1 Screen Size
2 Fit to Page

PASTE( )

PASTE

The PASTE function is the equivalent of the Paste command from the Edit menu.

PASTE.SPECIAL(parts,operation)
PASTE.SPECIAL?( )

PASTE.SPECIAL (Form 1)

This PASTE.SPECIAL function is the equivalent of the Paste Special command from the Edit menu, if you have copied a worksheet selection, and the document in the active window is a worksheet.

Parts must be a number:

1 All
2 Formulas
3 Values
4 Formats

Operation must be a number:

1 None
2 Add
3 Subtract
4 Multiply
5 Divide

PASTE.SPECIAL (Form 2)

PASTE.SPECIAL(values-in,series-names,categories)
PASTE.SPECIAL?( )

This PASTE.SPECIAL function is the equivalent of the Paste Special command from the Edit menu, if you have copied a worksheet selection, and the document in the active window is a chart.

Values-in must be a number:

1	Rows
2	Columns

Series-names and *categories* must be logical values:

TRUE	Checked
FALSE	Not checked

PASTE.SPECIAL (Form 3)

PASTE.SPECIAL(parts)
PASTE.SPECIAL?( )

This PASTE.SPECIAL function is the equivalent of the Paste Special command from the Edit menu, if you have selected and copied a chart, and the document in the active window is another chart.

Parts must be a number:

1	All
2	Formats
3	Formulas

PRECISION

PRECISION(logical)

The PRECISION function is the equivalent of the Precision As Displayed and Full Precision commands from the Options menu.

If *logical* is TRUE, Excel carries out the Full Precision command. If *logical* is FALSE, Excel carries out the Precision As Displayed command.

PREFERRED( )

PREFERRED

The PREFERRED function is the equivalent of the Preferred command from the Gallery menu.

PRINT(range,from,to,copies,feed)
PRINT( )
PRINT?( )

PRINT

The PRINT function is the equivalent of the Print command from the File menu.

If TTY is the printer specified in the Printer Setup command, or if Macintosh is specified and your printer is an ImageWriter™, ImageWriter15, or LaserWriter, you can specify the arguments *range*, *from*, *to*, *copies*, and *feed*.

If Macintosh is specified and your printer is not of these, you cannot specify any arguments. In this case, Excel uses the current settings in the Print command.

Range must be a number:

1 All

2 From

From, *to*, and *copies* must be numbers.

Feed must be a number:

1 Continuous

2 Cut Sheet

PRINTER.SETUP

PRINTER.SETUP(printer,port,baud)
PRINTER.SETUP?( )

The PRINTER.SETUP function is the equivalent of the Printer Setup command from the File menu.

Printer must be a number:

1	Macintosh
2	TTY

Port must be a number:

1	Printer
2	Modem

Baud must be a number:

1	300
2	600
3	1200
4	2400
5	4800
6	9600

PROTECT.DOCUMENT

PROTECT.DOCUMENT(logical)
PROTECT.DOCUMENT?( )

The PROTECT.DOCUMENT function is the equivalent of the Protect and Unprotect Document commands from the Options menu.

If *logical* is TRUE, Excel carries out the Protect Document command. If *logical* is FALSE, Excel carries out the Unprotect Document command.

To protect a document with a password, or to unprotect a document that has been protected with a password, use the form PROTECT.DOCUMENT?( ).

RELREF(ref-1,ref-2)

RELREF

The RELREF function gives the reference of *ref-1*, relative to the upper-left corner cell of *ref-2*. The reference is given as an R1C1-style relative reference in the form of text, such as "R[1]C[1]".

Examples

RELREF(A1,C3) *equals* "R[−2]C[−2]"

RELREF(Finance!A1,Finance!C3) *equals* "R[−2]C[−2]"

RELREF(A1:E5,C3:G7) *equals* RELREF(A1:E5,C3) *equals* "R[−2]C[−2]:R[2]C[2]"

REMOVE.PAGE.BREAK( )
SET.PAGE.BREAK( )

REMOVE.PAGE.BREAK
SET.PAGE.BREAK

The REMOVE.PAGE.BREAK and SET.PAGE.BREAK functions are the equivalents of the Remove and Set Page Break commands from the Options menu.

RESULT(type)

RESULT

The RESULT function allows you to specify the type of value to be returned by a function macro. If your function macro returns single values (either text, numbers, or logical values), the RESULT function is optional. To return an array or a reference, the RESULT function is mandatory.

The RESULT function, if used, must come before any other formulas in the macro, including the ARGUMENT functions.

Type determines what values Excel will return from a function macro. The value returned by the RETURN function should be of the specified type. If it is not, Excel first attempts to convert it to the specified type. If the value cannot be converted, the macro returns the error value #VALUE!.

Type must be a number:

1	Number
2	Text
4	Logical
8	Reference
16	Error
64	Array

Type can be the sum of the different numbers above, in which case the value returned can be of different types.

If you omit *type*, it is assumed to be 7, which is 1 + 2 + 4, and the value returned can be a single number, text, or logical value.

For more information on using the RESULT function, see "Passing Arguments and Returning Results" in Chapter 7, "More Information about Macro Sheets."

RETURN

RETURN(values)
RETURN( )

The RETURN function tells Excel to stop the currently running macro. If the macro was called with the Run command (or an Option-Command-key), all macro calculation stops and control is returned to the user. If the macro was called from a function on a worksheet or from another macro, calculation resumes at the formula containing the calling function. In that case, RETURN gives *values* as the result of the calling function.

Values can be a number, text, logical value, error value, array, or reference. Its type can be specified by the RESULT function.

RUN

RUN(ref)
RUN?( )

The RUN function is the equivalent of the Run command from the Macro menu.

Ref should be either an external reference to a macro on a macro sheet, such as Macros!$A$1 or Macros!Months, or an R1C1-style external reference to a macro in the form of text, such as "Macros!R1C1" or "Macros!Months".

If you are recording a macro, the reference you enter in the Run dialog box is recorded as text, with A1-style references converted to R1C1-style references.

Examples

Each of the following functions runs the macro beginning at the upper-left corner of the range named Macros on the macro sheet named Months:

RUN(Macros!Months)
RUN("Macros!Months")

SAVE( )

SAVE

The SAVE function is the equivalent of the Save command from the File menu.

SAVE.AS(document,type)
SAVE.AS?( )

SAVE.AS

The SAVE.AS function is the equivalent of the Save As command from the File menu.

Document must be the name of a document in the form of text, such as "Sales".

Type does not apply if the document in the active window is a chart.

Type must be a number:

1	Normal
2	SYLK
3	Text
4	WKS

You can save a worksheet as any of types 1 through 4.

SAVE.AS always saves the specified document on the disk in the default disk drive. To save the document on another disk or another disk drive, use the form SAVE.AS?( ).

SELECT

SELECT(selection-ref,active-cell-ref)

The SELECT function is equivalent to the action of selecting cells or changing the active cell on the active worksheet. It selects the cells specified by *selection-ref* and makes the cell specified by *active-cell-ref* the active cell.

Selection-ref should be either a reference to the active worksheet, such as !A1A3 or !Sales, or an R1C1-style relative reference in the form of text, such as "R[−1]C[−1]:R[1]C[1]". If *selection-ref* is a reference, SELECT selects that reference. If *selection-ref* is a relative reference in the form of text, it is assumed to be relative to the active cell in the current selection. If you omit *selection-ref*, SELECT does not change the selection.

Active-cell-ref should be either a reference to a single cell on the active worksheet, such as !A1, or an R1C1-style relative reference to a single cell in the form of text, such as "R[−1]C[−1]". If *active-cell-ref* is a reference, SELECT makes that cell the active cell. *Active-cell-ref* must be inside *selection-ref*. If *active-cell-ref* is a relative reference in the form of text, it is assumed to be relative to the active cell in the current selection. If you omit *active-cell-ref*, SELECT makes the upper-left corner cell in *selection-ref* the active cell.

If you are recording a macro with Relative Record on, and you make a selection, Excel records the action using R1C1-style relative references in the form of text. If you are recording with Absolute Record on, Excel records the action using absolute references.

Examples

This function selects cells C3:E5 on the active worksheet and makes C3 the active cell:

```
SELECT(!C3:E5,!C3)
```

If the active cell is C3, this function selects cells E5:G7 and makes F6 the active cell:

```
SELECT("R[2]C[2]:R[4]C[4]","R[3]C[3]")
```

The following sequence of functions:

SELECT(,"RC[1]")
SELECT(,"RC[−1]")
SELECT(,"R[1]C")
SELECT(,"R[−1]C")

moves the active cell within a range or multiple selection to the right, to the left, down, and up within the selection, just as the Tab, Shift-Tab, Return, and Shift-Return keys do.

SELECT.CHART( )

SELECT.CHART

The SELECT.CHART function is the equivalent of the Select Chart command from the Chart menu.

SELECT.LAST.CELL( )

SELECT.LAST.CELL

The SELECT.LAST.CELL function is the equivalent of the Select Last Cell command from the Formula menu.

SELECTION( )

SELECTION

The SELECTION function gives the reference of the selection as an external reference.

See also ACTIVE.CELL.

Example

If the document in the active window is named Sales, and if A1:A3 is the selection, then:

SELECTION( ) *equals* Sales!A1:A3

SET.CRITERIA( )

SET.CRITERIA

The SET.CRITERIA function is the equivalent of the Set Criteria command from the Data menu.

SET.DATABASE

SET.DATABASE( )

The SET.DATABASE function is the equivalent of the Set Database command from the Data menu.

SET.NAME

SET.NAME(name,value)

The SET.NAME function defines a name on the macro sheet to refer to the specified value. The SET.NAME function is useful for storing values during the calculation of a macro.

Name must be text.

Value can be a number, text, logical value, error value, array, or reference. If *value* is a reference, *name* is defined to refer to that reference. If you want to define *name* to refer to the value of a referenced cell rather than to the reference itself, you must use the DEREF function.

Examples

This function defines the name I to refer to the constant number 1 on the macro sheet:

```
SET.NAME("I",1)
```

And if the value of I is 1, this function defines I to refer to 2:

```
SET.NAME("I",I+1)
```

This function defines the name Results to refer to the cells A1:C3:

```
SET.NAME("Results",A1:C3)
```

If cell A1 contains the value 2, this function defines the name Reference to refer to cell A1:

```
SET.NAME("Reference",A1)
```

And this function defines the name Value to refer to the value 2:

```
SET.NAME("Value",DEREF(A1))
```

SET.PAGE.BREAK( )

SET.PAGE.BREAK

See REMOVE.PAGE.BREAK.

SET.PRINT.AREA( )

SET.PRINT.AREA

The SET.PRINT.AREA function is the equivalent of the Set Print Area command from the Options menu.

SET.PRINT.TITLES( )

SET.PRINT.TITLES

The SET.PRINT.TITLES function is the equivalent of the Set Print Titles command from the Options menu.

SET.VALUE(ref,values)

SET.VALUE

The SET.VALUE function changes the value of the cells specified by *ref* to *values*. If a cell previously contained a formula, the formula is not changed. SET.VALUE is useful for storing values and for looping during the calculation of a macro.

Ref must be a reference to cells on the macro sheet. If *ref* is a reference to a range of cells, rather than to a single cell, then *values* should be an array of the same size. If not, Excel expands it to that size using the normal rules for expanding arrays.

Examples

This function changes the value of cell A1 on the macro sheet to 1:

```
SET.VALUE(A1,1)
```

If that cell contains the formula =A1+1, the next time that formula is calculated, the value of the cell will become 2.

This function stores the values 1, 2, 3, and 4 in cells A1, B1, A2, and B2:

```
SET.VALUE(A1:B2,{1,2;3,4})
```

SHOW.ACTIVE.CELL

SHOW.ACTIVE.CELL( )

The SHOW.ACTIVE.CELL function is the equivalent of the Show Active Cell command from the Formula menu.

SHOW.CLIPBOARD

SHOW.CLIPBOARD( )

The SHOW.CLIPBOARD function is the equivalent of the Show Clipboard command from the Window menu.

SIZE

SIZE(x-number,y-number,window)

The SIZE function is equivalent to the action of changing the size of a window by dragging its size box. It moves the lower-right corner of the specified window so that it has the width and height specified by *x-number* and *y-number*. SIZE does not change the position of the upper-left corner of the window or affect whether the specified window is active or inactive.

X-number is the width of the window. The width of the screen is 512. *Y-number* is the height of the window. The height of the screen is 342. *Y-number* must be smaller than 303, which is the height of the screen from the bottom of the formula bar to the bottom of the screen.

Window must be the name of a window in the form of text, such as "Sales", or "Sales:1" if there is more than one window displaying the document named Sales. If *window* is "Clipboard", Excel changes the size of the Clipboard. If you omit *window*, Excel changes the size of the active window.

See also MOVE.

Examples

This function changes the size of the active window so that its width is 250, approximately half the width of the screen, and its height is 150, approximately half the height of the screen:

```
SIZE(250,150)
```

This function does the same thing to the document named Sales:

```
SIZE(250,150,"Sales")
```

SORT(sort-by,1st-key,order,2nd-key,order,3rd-key,order)
SORT?( )

SORT

The SORT function is the equivalent of the Sort command from the Data menu.

Sort-by must be a number:

1 Rows

2 Columns

Order must be a number:

1 Ascending

2 Descending

1st-key, *2nd-key*, and *3rd-key* should be either external references to the active worksheet, such as !B:B or !Price, or R1C1-style references in the form of text, such as "C2" or "C[1]", or "Price". If the keys are R1C1-style relative references in the form of text, they are assumed to be relative to the active cell in the selection.

If you are recording a macro, the keys you specify are recorded as text, with A1-style references converted to R1C1-style references. If you want the keys recorded relative to the active cell in the selection, you should enter relative references in the Sort command dialog box. If you want the keys recorded absolutely, you should enter absolute references.

Because the keys may be either normal references or references in the form of text, you may sometimes want to use the DEREF function. See the examples below.

Examples

Each of the following functions sorts the selection by rows using column C as the key in ascending order:

```
SORT(1,!$C:$C,1)
SORT(1,"C3",1)
```

If the selection is C1:E50 and the active cell is C1, this function sorts the selection by rows using column E, the third column in the selection, as the first key in descending order, and column D, the second column in the selection, as the second key in ascending order:

```
SORT(1,"C[2]",2,"C[1]",1)
```

If column D is named Amount, each of the following functions sorts the selection by rows using column D as the key in descending order:

```
SORT(1,!Amount,2)
SORT(1,"Amount",2)
```

If cell B2 on the active worksheet contains the text "Amount", the first function below sorts the selection using column B as the key, while the second function below sorts the selection using the column named Amount as the key:

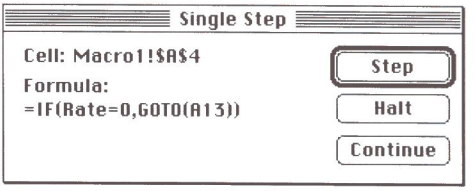
```
SORT(1,!$B$2,2)
SORT(1,DEREF(!$B$2),2)
```

STEP

STEP( )

The STEP function allows you to calculate a macro one cell at a time. This is a useful debugging tool.

When Excel encounters a STEP function, it stops macro calculation and displays a dialog box that tells you which cell in the macro Excel is about to calculate, and the formula in that cell.



The dialog box contains Step, Halt, and Continue buttons. If you click Step or press the Enter key, Excel calculates the next cell in the macro and stops again. The dialog box remains on the screen so that you can then click Step or press Enter again. If you click Halt, Excel halts the macro. If you click Continue or press Command-.(period), Excel continues normal calculation.

The dialog box always displays the reference of the next cell to be calculated. You can move the dialog box by dragging its title bar. This allows you to examine your screen while the macro is stopped, waiting for you to click a button or press a key.

You can also turn single-stepping on and off while a macro is running. If you press Command-.(period) while a macro is running, Excel displays a dialog box with Step, Halt, and Continue buttons. If you click the Step button, Excel displays the single-stepping dialog box described above.

STYLE(bold,italic)
STYLE?( )

STYLE

The STYLE function is the equivalent of the Style command from the Format menu.

Bold and *italic* must be logical values:

TRUE	Checked
FALSE	Not checked

TABLE(row-input,column-input)
TABLE?( )

TABLE

The TABLE function is the equivalent of the Table command from the Data menu.

Row-input and *column-input* should be either external references to single cells on the active worksheet, such as !$A$1 or !Price, or R1C1-style references to single cells in the form of text, such as "R1C1", "R[−1]C[−1]", or "Price". If *row-input* or *column-input* are R1C1-style references, they are assumed to be relative to the active cell in the selection.

If you are recording a macro, the input cells you enter are recorded as text, with A1-style references converted to R1C1-style references. If you want the input cells recorded relative to the active cell in the selection, you should enter relative references in the Table dialog box. If you want the input cells recorded absolutely, you should enter absolute references.

Because *row-input* and *column-input* can be either normal references or references in the form of text, you may sometimes want to use the DEREF function. See the examples below.

Examples

Each of the following functions creates a two-input table in the selection on the active sheet using A1 and A2 as input cells:

TABLE(!$A$1,!$A$2)
TABLE("R1C1","R2C2")

If the selection is C3:E5 and the active cell is C3, this function creates a one-input table in the selection using B2 as the column-input cell:

TABLE(,"R[−1]C[−1]")

If cell B2 on the active worksheet is named Size, each of the following functions creates a one-input table in the selection using B2 as the row-input cell:

TABLE(!Size,)
TABLE("Size",)

If cell B2 on the active worksheet contains the text "Size", the first function below sets up a one-input table in the selection using B2 as the row-input cell, while the second function below sets up a one-input table using the cell named Size as the row-input cell.

TABLE(!$B$2,)
TABLE(DEREF(!$B$2),)

UNDO

UNDO( )

The UNDO function is the equivalent of the Undo command from the Edit menu.

UNLOCKED.NEXT
UNLOCKED.PREV

UNLOCKED.NEXT( )
UNLOCKED.PREV( )

The UNLOCKED.NEXT and UNLOCKED.PREV functions are equivalent to the actions of pressing Enter or Shift-Enter to move the active cell to the next or previous unlocked cell in a protected worksheet.

VLINE(number)

See HLINE.

VPAGE(number)

See HPAGE.

VSCROLL(row-number)

See HSCROLL.

WINDOWS( )

The WINDOWS function gives, as an array of text values, the names of all windows on the screen. The names are given in order of their level on the screen, the first name being that of the active window. With the INDEX function, you can select individual window names from the array for use in other functions that take window names as arguments.

Example

If the active window, named Budget, is on top of a window named Actual:2, which is on top of a window named Actual:1, then:

WINDOWS( ) *equals* {"Budget","Actual:2","Actual:1"}

Index

A1.R1C1 function 140, 144
ABS function 33, 39
Absolute Record command 117
Absolute reference recording 117
Absolute value 33, 39
ABSREF function 143, 144
ACOS function 35, 39
Action-equivalent function 82, 141-142
Action-taking formula 86, 127-128
Action-taking function
  action-equivalent function *See* Action-equivalent function
  calculation procedure 127-128
  command-equivalent function *See* Command-equivalent function
  command macro component 91
  description 82
  entry 127
  failure 128
  formulas 86, 127-128
  INPUT function combination 128
  number per formula 128
  question-mark form *See* Command-equivalent function
  results 82, 127-128
ACTIVATE function 141, 145
ACTIVATE.NEXT function 141, 145
ACTIVATE.PREV function 141, 145
Active cell
  changing 142, 182-183
  formula entry 160
  function *See* ACTIVE.CELL function
  moving to unlocked cell 142
  reference 143, 145
  showing 186
Active worksheet 117-118
ACTIVE.CELL function 143, 145
Alarm *See* Bell
Alert
  array operations 12
  beep 148
Alert *(continued)*
  display 142, 146
  function *See* ALERT function
  function entry 26
  input type error 168
  macro sheet closed 123
  recorder range 116
  types 146
ALERT function 97, 142, 146
ALIGNMENT function 140, 147
AND function 32, 36, 39
Arccosine *See* ACOS function
Arcsine *See* ASIN function
Arctangent *See* ATAN function; ATAN2 function
Area chart *See* GALLERY.AREA function
AREAS function 32, 38, 40
Argument
  See also specific function
  array *See* Array argument
  command-equivalent function 137-138
  constants 27
  defined 26
  description 30
  enclosure 26, 105
  entry in cells 112, 133-134
  formulas 27
  function *See* ARGUMENT function
  function macro 105, 111-112, 136
  multiple-value *See* Array argument
  naming 108, 111-112, 132-133
  nesting 27
  number of 29
  omission, value 132
  passing 132-136, 142, 144, 147-148
  reference *See* Reference argument
  replacement 96
  single-value 5, 31
  text enclosure 31
  type
    array *See* Array argument
    conversion 111, 134, 136

Argument *(continued)*
type *(continued)*
default 135
description 30
reference *See* Reference argument
specification 111, 134-135, 136
vector 29, 30
ARGUMENT function
argument
entry in cell 112, 133-134
naming 108, 111, 132, 133
passing 132-136, 142, 147-148
type specification 111-112, 134-135
value entry 112
description 147-148
forms 132, 133
location 132, 147
references 133-134, 147
Array
argument *See* Array argument
charts 18-19
columns 38, 42
constant 11-12, 19
defined 7
equal size 13-16
expansion 13-16
external reference *See* External reference
formula *See* Array formula
results *See* Array result
size 13-16, 134
transposing 72
Array argument
array functions 32-33
equal-sized arrays 13-16
expansion 13-16
function macro 113-114, 136
naming 133
specification 134-135
type 29, 30
values changing 133
Array constant 11-12, 19
Array formula
calculating 124
clearing 12
cutting 12
defined 8
deleting 12
editing 12
entry 8, 9, 141, 160
evaluating 12
Array formula *(continued)*
external references 17-18
formatting 12
function macro 113
relative reference 12
Array function 16, 32-33
Array result
array formula 8-10, 14-16
array functions 16, 31, 32-33
equal size 14-16
function macro 113-114, 179-180
specification 135, 179-180
type number 73, 180
ASIN function 35, 40
Asterisk (*)
alert box icon 146
TEXT function restriction 71
wildcard character 59, 69
ATAN function 35, 40
ATAN2 function 35, 41
AVERAGE function
arguments 32, 41
description 27, 41
statistical function 34

Bar chart *See* GALLERY.BAR function
BEEP function 142, 148
Bell 142, 148
Bold style 189
BORDER function 140, 149
Braces ({})
array constant enclosure 11
array formula enclosure 8
Branching
calling method *See* Calling a macro
GOTO function method 98-99, 116, 125, 142, 165
looping *See* Looping
Built-in function *See* Worksheet function

CALCULATE.NOW function 140, 149
Calculation
action-taking functions 127-128
array formulas 124
branching *See* Branching
calling a macro *See* Calling a macro
conditionals 129-130
constant value effect 119, 124

Calculation *(continued)*
control 129-130
formula location 124-125, 129
function *See* CALCULATE.NOW function; CALCULATION function
halting *See* stopping
interrupting 131
looping *See* Looping
macro path 125, 129
macro sheet/worksheet differences 83, 98, 123
order 89, 98-100, 124-126
single-step 131, 143, 188-189
step-by-step *See* single step
stopping 98, 127, 131, 143, 165-166, 180
CALCULATION function 140, 149
CALLER function 143, 150
Calling a macro
calculation order 99-100, 125-126
calling function reference 143, 150
calling macro defined 127
command macros 121
example 107-109
external references 121-122
function form 126
function macros 107-109, 121
macro sheet open 123
methods 121
nested macro defined 127
ref 99, 126-127, 142, 144
references 121-122
CANCEL.COPY function 141, 152
Caret (^), exponentiation operator 48
Cash flow *See* FV function; PV function
Cell
active *See* Active cell
argument entry 133-134
array entry *See* Array
contents 124
finding 142, 161
last cell 183
linking 17
naming 120
protection 140, 150
range *See* Cell range
recorder range setting 87, 115
searching 142, 161
selecting 87, 142, 182
value change 143, 185
Cell range
array formula entry 9, 12, 113-114
consolidating 18
linking 17
recorder range 87, 115
CELL.PROTECTION function 140, 150
Chart
arrays 18-19
copying 152
function *See* COMBINATION function; COPY.CHART function; GALLERY.AREA function; GALLERY.BAR function; GALLERY.COLUMN function; GALLERY.LINE function; GALLERY.PIE function; GALLERY.SCATTER function; LEGEND function; MAIN.CHART.TYPE function; OVERLAY.CHART.TYPE function; PREFERRED function; SELECT.CHART function
legend 169
main type 170
overlay type 174
selection 183
Chart menu 141
CHOOSE function 36, 42
CLEAR function 139, 150
Clearing array formula 12
Clipboard
contents 186
moving 171
picture copying 151
sizing 186
CLOSE function 141, 151
Column
arrays 38, 42
chart *See* GALLERY.COLUMN function
width control 141, 151-152
COLUMN function 32, 38, 42
COLUMNS function
arguments 32
array size determination 134
description 42
smart macros 94-95
special-purpose function 38
COLUMN.WIDTH function 140, 141, 151-152
COMBINATION function 141, 152
Comma (,)
argument separation 26, 105
array value separation 11

Command
  Absolute Record 117
  choice recording 87
  Define Name
    command macro naming 88, 92
    function macro naming 105
    macro sheet options 83, 88, 122-123
    name listing 133
  Delete 93
  function equivalent *See* Command-equivalent function
  New 83
  Paste Function
    function entry 26, 27
    function macro entry 106
    functions listing 83
  Relative Record 117
  Run
    command macro calling 121
    command macro running 85, 89, 92
  Set Recorder
    availability 83
    recorder range setting 87, 90, 115
  Start Recorder 87, 90, 116
  Stop Recorder 87, 90, 116
Command-.(period) 131, 189
Command-Enter 8
Command-equivalent function
  arguments 137-138
  Chart menu functions 141
  Data menu functions 140
  description 82
  Edit menu functions 139
  File menu functions 138-139
  Format menu functions 140
  Formula menu functions 139
  Gallery menu functions 141
  interactive command macro 85, 95-98, 138
  Macro menu functions 141
  Options menu functions 140
  question-mark form 96, 97, 138
  Window menu functions 140
Command-equivalent macro 100-101
Command-H 161
Command key
  array formula editing 12
  keyboard commands *See* Keyboard command
  Option key combination *See* Option-Command-key
Command macro
  argument passing *See* Argument
  calculation *See* Calculation
  calling *See* Calling a macro
  creating 90-92
  defined 81, 85
  editing 93-94
  entry 86, 90
  examples 86, 90-92
  interactive 85, 95-98, 138
  naming 88, 92
  purpose 81
  question-mark form 96, 97, 138
  recorder range setting 115
  recording
    absolute references 117
    efficiency 87
    procedure 86-87, 90-91, 115-117
    relative references 117
  running 85, 89, 92
  sample 90-92
  smart 94-95
  stopping 89, 98
Command-Shift-H 161
Comment 110, 120-121
Comparison value 58-59
Conditionals *See* Calculation
Constant array 11-12, 19
Constant value
  argument, use as 27
  array constants 11-12, 19
  calculation, effect 119, 124
  formula, entry 11
Control functions 142-143
COPY function 139, 152
COPY.CHART function 139, 152
COPY.PICTURE function 139, 152
COS function 35, 43
COUNT function
  arguments 32, 43
  description 43
  statistical function 34
Counter *See* Looping
CREATE.NAMES function 139, 153
Criteria
  external reference prohibition 44
  setting 183
CUT function 139, 153
Cutting array formula 12

Data menu 140
Database
  external reference prohibition 44
  finding 153
  setting 184
  worksheet functions 35
DATA.DELETE function 140, 153
DATA.FIND function 140, 153
DATA.FIND.NEXT function 141, 153
DATA.FIND.PREV function 141, 153
DATA.SERIES function 140, 154
DATE function 37, 43-44
Date functions 37-38
DAVERAGE function 35, 44-45
DAY function 37, 45
DCOUNT function 35, 46
Debugging 131, 188
Define Name command
  command macro naming 88, 92
  function macro naming 105
  macro sheet options 83, 88, 122-123
  name listing 133
DEFINE.NAME function 139, 154-155
Delete command 93
DELETE.FORMAT function 140, 156
DELETE.NAME function 139, 156
Deleting
  array formula 12
  data 153
  document 159
  EDIT.DELETE function 157
  format 156
  macro 118
  name 156
Depreciation function macro 113-114
DEREF function 143, 156
Dialog box
  action-taking function failure 128
  Alignment command 138
  Define Name command 88, 105
  interactive command macro 95, 96, 138
  Paste Function command 106
  Run command 89
  single step 131
DISPLAY function 140, 156
#DIV/0!
  ATAN2 function error 41
  ISERROR function value 54
  MOD function error 62
DMAX function 35, 46
DMIN function 35, 46
Document
  deleting 159
  opening
    functions 173
    listing 143, 157
  protection 178
  saving 181
Documentation 119-121
DOCUMENTS function 143, 157
DOLLAR function 36, 47
Double quotation marks (" ") 11, 31
DSTDEV function 35, 47
DSUM function 35, 47
DVAR function 35, 48

ECHO function 142, 157
Edit menu 139
EDIT.DELETE function 139, 157
Editing
  array formula 12
  command macro 93
Error check 142, 158
ERROR function 142, 158
Error value
  See also specific error value
  action-taking function 82, 127
  array constant component 11
  type number 73
Excel Macro Language *See* Macro Language defined
Exclamation point (!)
  alert box icon 146
  external reference component 118, 121-122
EXP function 33, 48
Exponential function *See* EXP function
Exponentiation operator 48
External reference
  active cell 145
  active worksheet 117-118
  array formulas 17-18
  column widths 151
  criteria prohibition 44
  database prohibition 44
  description 121-122
  linking 17
  macro calling 121-122, 144
  master worksheet creation 18
  passing 136

External reference *(continued)*
reference argument 136
SELECTION function 183
updating 173
value consolidation 18
EXTRACT function 140, 158

FALSE function 32, 36, 48
FALSE value *See* Logical value
File menu 138-139
FILE.DELETE function 139, 159
FILL.DOWN function 139, 159
FILL.RIGHT function 139, 159
Financial functions 37
Finding
cells 142, 161
database 153
matching records 141, 153
text 37, 68-69
FIXED function 36, 49
FONT function 140, 159
Footer 174
FOR-NEXT loop 130
Format
array formula 12
deleting 156
number 36, 49, 71, 159
Format menu 140
FORMAT.NUMBER function 140, 159
Formula
action-taking function 86, 127-128
array *See* Array formula
bar 143, 164
calculation, location 124-125, 129
cell reference 18
chart *See* Series formula
constants entry 11
deleting 93
entry 87, 141, 142, 160-161
function entry 27
function macro entry 104, 106
macro components 81, 103, 124
macro sheet, entry 86-87
multiple arguments *See* Array
series formula 18-19
Formula bar 143, 164
FORMULA function 141, 160
Formula menu 139
FORMULA.ARRAY function 141, 160
FORMULA.FILL function 142, 161
FORMULA.FIND function 139, 161
FORMULA.FIND.NEXT function 142, 161
FORMULA.FIND.PREV function 142, 161
FORMULA.GOTO function 139, 162
FULL function 142, 162
Function
See also specific function
action-taking *See* Action-taking function
argument *See* Argument
built-in *See* Worksheet function
command-equivalent *See* Command-equivalent function
contents 26
database 35
date 37-38
defined 25
entry 27
financial 37
logical 36
macro calling *See* Calling a macro
Macro Language *See* Macro Language function
mathematical 33-34
nesting 27
pasting 26, 27
question-mark form 97, 138
result *See* Result
special-purpose 38, 143
statistical 34
syntax 26
text 36-37
trigonometric 35-36
worksheet *See* Worksheet function
Function macro
arguments *See* Argument
array argument 136
array results 113-114
calculation *See* Calculation
calling *See* Calling a macro
comments 110, 120-121
components 103
creating 103-104, 106-107, 109
defined 81, 103
enhancing 109-110
entry 104, 106
example 104, 106-109
FV function equivalent 106-110
improvement 109-110
information requirements 105

Function macro *(continued)*
looping *See* Looping
naming 105
purpose 81
recorder ineffective 103
results *See* Result
running 121-123
sample 106-109
stopping *See* Macro
using 105-106
writing 106-110
Future value formula 106-110
FV function
arguments 31, 65-66, 106, 109
description 49, 65-66, 106
financial function 37, 106-110

Gallery menu 141
GALLERY.AREA function 141, 163
GALLERY.BAR function 141, 163
GALLERY.COLUMN function 141, 163
GALLERY.LINE function 141, 163
GALLERY.PIE function 141, 163
GALLERY.SCATTER function 141, 164
GET.FORMULA function 143, 164
GET.NAME function 143, 164-165
GOTO function
calculation branching 98-99, 116, 125, 142, 165
description 165
destination label 119-120
recorder range extension 116
GROWTH function 32, 34, 49

HALT function
calculation stopping 98, 127, 143, 165-166
description 165-166
Header 174
HLINE function 142, 166
HLOOKUP function 38, 50
HOUR function 37, 50-51
HPAGE function 142, 166
HSCROLL function 142, 166-167

IF function 36, 51, 94-95
IF-THEN-ELSE condition 129
INDEX function
arguments 32
array values 133
description 51-53
function macro use 114
reference results 136
special-purpose function 38
INPUT function
action-taking function combination 128
description 143, 167-169
interactive command macro creating 96-97
INSERT function 139, 169
INT function 33, 53
Integer function *See* INT function
Interaction control functions 142-143
Interactive command macro 85, 95-98, 138, 167-169
Interest rate *See* RATE function
Investments *See* FV function
IRR function 37, 53-54
ISERROR function 36, 54
ISNA function 36, 54
ISREF function 32, 36, 55
Italics style 189
Iteration *See* IRR function; RATE function

Keyboard command
Command-.(period) 131, 189
Command-Enter 8
Command-H 161
Command-Shift-H 161

Label 119-121
LEGEND function 141, 169
LEN function 27, 36, 55
LINEST function
arguments 32
description 55-56
result size 16
statistical function 34
Linking
external references use 17
functions *See* LINKS function
linked document listing 173
supporting worksheets listing 169-170
LINKS function 143, 169-170
LN function 34, 56
LOG10 function 34, 56

Logarithm *See* LN function; LOG10 function; LOGEST function
LOGEST function 32, 34, 57
Logical functions 36
Logical value
  action-taking function result 82, 127-128
  argument type 29, 30
  array constant 11
  type number 73
LOOKUP function 38, 57-58
Looping
  description 101
  FOR-NEXT 130
  function macro 113
  loop counter setup 101, 129
  WHILE-DO 130

Macro
  argument *See* Argument
  calculation *See* Calculation
  calling macro *See* Calling a macro
  cell
    contents 124
    naming 120
  command macro *See* Command macro
  comments 110, 120-121
  components 81, 103, 124
  defined 81
  deleting 118
  documentation 119-121
  ending *See* stopping
  function macro *See* Function macro
  functions *See* Macro Language function
  halting *See* stopping
  interactive 85, 95-98, 138, 167-169
  interrupting 131
  labels 119-121
  looping *See* Looping
  naming 119-120, 122
  nesting 127
  path 125, 129
  recording *See* Recording
  result *See* Result
  running 85, 89, 92, 121-123, 180-181
  sheet *See* Macro sheet
  single step 131, 143, 188-189
  stopping 89, 98, 116, 124, 127, 131, 143, 165-166, 180
  storing 83, 118

Macro *(continued)*
  types 81
Macro Language defined 81, 82
Macro Language function
  A1.R1C1 140, 144
  ABSREF 143, 144
  action-equivalent *See* Action-equivalent function
  action-taking *See* Action-taking function
  ACTIVATE 141, 145
  ACTIVATE.NEXT 141, 145
  ACTIVATE.PREV 141, 145
  ACTIVE.CELL 143, 145
  ALERT 97, 142, 146
  ALIGNMENT 140, 147
  ARGUMENT
    argument entry in cell 112, 133-134
    argument naming 108, 111, 132, 133
    argument passing 132-136, 142, 147-148
    argument type specification 111-112, 134-135
    argument value entry 112
    description 147-148
    forms 132, 133
    location 132, 147
    references 133-134, 147
  arguments
    See also Argument
    description 137-138
  BEEP 142, 148
  BORDER 140, 149
  CALCULATE.NOW 140, 149
  CALCULATION 140, 149
  CALLER 143, 150
  CANCEL.COPY 141, 152
  categories 138-143
  CELL.PROTECTION 140, 150
  CLEAR 139, 150
  CLOSE 141, 151
  COLUMN.WIDTH 140, 141, 151-152
  COMBINATION 141, 152
  command-equivalent *See* Command-equivalent function
  COPY 139, 152
  COPY.CHART 139, 152
  COPY.PICTURE 139, 152
  CREATE.NAMES 139, 153
  CUT 139, 153
  DATA.DELETE 140, 153
  DATA.FIND 140, 153

Macro Language function *(continued)*
DATA.FIND.NEXT 141, 153
DATA.FIND.PREV 141, 153
DATA.SERIES 140, 154
DEFINE.NAME 139, 154-155
DELETE.FORMAT 140, 156
DELETE.NAME 139, 156
DEREF 143, 156
description 82
DISPLAY 140, 156
DOCUMENTS 143, 157
ECHO 142, 157
EDIT.DELETE 139, 157
ERROR 142, 158
EXTRACT 140, 158
FILE.DELETE 139, 159
FILL.DOWN 139, 159
FILL.RIGHT 139, 159
FONT 140, 159
FORMAT.NUMBER 140, 159
FORMULA 141, 160
FORMULA.ARRAY 141, 160
FORMULA.FILL 142, 161
FORMULA.FIND 139, 161
FORMULA.FIND.NEXT 142, 161
FORMULA.FIND.PREV 142, 161
FORMULA.GOTO 139, 162
FULL 142, 162
GALLERY.AREA 141, 163
GALLERY.BAR 141, 163
GALLERY.COLUMN 141, 163
GALLERY.LINE 141, 163
GALLERY.PIE 141, 163
GALLERY.SCATTER 141, 164
GET.FORMULA 143, 164
GET.NAME 143, 164-165
GOTO
  calculation branching 98-99, 116, 125, 142, 165
  description 165
  destination label 119-120
  recorder range extension 116
HALT
  calculation stopping 98, 127, 143, 165-166
  description 165-166
HLINE 142, 166
HPAGE 142, 166
HSCROLL 142, 166-167
INPUT 96-97, 128, 143, 167-169
INSERT 139, 169

Macro Language function *(continued)*
LEGEND 141, 169
LINKS 143, 169-170
MAIN.CHART.TYPE 141, 170
MESSAGE 143, 170-171
MOVE 142, 171
NEW 138, 172
NEW.WINDOW 140, 172
OFFSET 114, 136, 143, 172
OPEN 138, 173
OPEN.LINKS 138, 173
OVERLAY.CHART.TYPE 141, 174
PAGE.SETUP 139, 174-175
PASTE 139, 175
PASTE.SPECIAL 139, 175-176
PRECISION 140, 176
PREFERRED 141, 177
PRINT 139, 177
PRINTER.SETUP 139, 178
PROTECT.DOCUMENT 140, 178
ref
  ARGUMENT function 133-134
  cell value 143, 156
  description 144
  GOTO function 98, 125, 142, 165
  macro calling 99, 126-127, 142, 144
  offset reference 143, 172
  relationships 143, 144, 179
RELREF 143, 179
REMOVE.PAGE.BREAK 140, 179
RESULT
  description 143, 179-180
  form 135
  location 132, 135, 179
  omission 112, 135
  result type specification 111, 135, 143, 179-180
RETURN
  description 180
  macro stopping 89, 98, 116, 124, 127, 143, 180
  overwrite 116
  value array return 114
RUN 141, 180-181
SAVE 139, 181
SAVE.AS 139, 181
SELECT 142, 182-183
SELECT.CHART 141, 183
SELECTION 94-95, 143, 183
SELECT.LAST.CELL 139, 183

Macro Language function *(continued)*
SET.CRITERIA 140, 183
SET.DATABASE 140, 184
SET.NAME 110, 129, 143, 184
SET.PAGE.BREAK 140, 179
SET.PRINT.AREA 140, 185
SET.PRINT.TITLES 140, 185
SET.VALUE 112, 129, 133, 143, 185
SHOW.ACTIVE.CELL 139, 186
SHOW.CLIPBOARD 140, 186
SIZE 142, 186
smart macros 94-95
SORT 140, 187-188
STEP 143, 188-189
STYLE 140, 189
TABLE 140, 189-190
UNDO 139, 190
UNLOCKED.NEXT 142, 191
UNLOCKED.PREV 142, 191
VLINE 142, 166
VPAGE 142, 166
VSCROLL 142, 166-167
WINDOWS 143, 191
Macro menu 141
Macro sheet
active worksheet reference 117-118
calculation *See* Calculation
command macro entry 86-87
creating 83
defined 83
function macro entry 104
memory conservation 118
name
conventions 123
definition 143, 184
organization 118
recorder range *See* Recorder range
value change 143, 185
worksheet differences 83, 91, 98, 123
MAIN.CHART.TYPE function 141, 170
Master worksheet 18
MATCH function 38, 58-59
Matching records 141, 153
Mathematical functions 33-34
MAX function 32, 34, 60
Memory conservation 118
Message display 143, 170-171
MESSAGE function 143, 170-171
MID function 36, 60
MIN function 32, 34, 61
MINUTE function 37, 50-51
MIRR function 31, 37, 61-62
MOD function 34, 62
MONTH function 37, 45
MOVE function 142, 171

#N/A
argument value 132, 147
array argument expansion 13
HLOOKUP function error 50
ISERROR function value 54
ISNA function value 54
LINKS function error 169
LOOKUP function error 57, 58
MATCH function error 58, 59
NA function error 38, 62
NA function 32, 38, 62
Name
See also Label
argument 108, 111-112, 132-133
array argument 133
cells 120
command macro 88, 92
definition 143, 154-155, 164-165, 184
deleting 156
documents 143, 157
function *See* DEFINE.NAME function; DELETE.NAME function; GET.NAME function; SET.NAME function
function macro 105
GOTO function destination 119-120
macro 119-120, 122
macro sheet 123, 143, 184
precautions 120
restrictions 120
windows 143, 191
#NAME? 54
Natural logarithm *See* LN function
Nesting 27, 127
New command 83
NEW function 138, 172
NEW.WINDOW function 140, 172
NOT function 36, 63
NOW function 32, 37, 63
NPER function 37, 63, 65-66
NPV function 37, 63-64
#NULL! 54
#NUM!
GROWTH function error 49

#NUM! *(continued)*
IRR function error 53
ISERROR function value 54
LOGEST function error 57
RATE function error 65
SQRT function error 70
Number
absolute value 33, 39
argument type 29, 30
array constant 11
average *See* AVERAGE function; DAVERAGE function
conversion to text 37, 71
format 36, 49, 71, 159
random 34, 67
rounding off 34, 36, 47, 49, 67
serial number 43-44, 71-72
square root 34, 70
summing *See* DSUM function; SUM function
type number 73

OFFSET function
description 172
function macro use 114
reference argument 136
special-purpose function 143
OPEN function 138, 173
OPEN.LINKS function 138, 173
Operator
array argument 8
exponentiation 48
Option-Command-key
command macro
calling 121
running 85, 88, 92
setting 88, 92, 122-123
Options menu 140
OR function 33, 36, 64
OVERLAY.CHART.TYPE function 141, 174

Page breaks 179
PAGE.SETUP function 139, 174-175
Parentheses ( ), argument enclosure 26, 105
Passing an argument *See* Argument
PASTE function 139, 175
Paste Function command
function entry 26, 27
function macro entry 106
Paste Function command *(continued)*
functions listing 83
PASTE.SPECIAL function 139, 175-176
PI function 33, 34, 64
Picture copying 152
Pie chart *See* GALLERY.PIE function
PMT function 37, 65-66
PRECISION function 140, 176
PREFERRED function 141, 177
PRINT function 139, 177
PRINTER.SETUP function 139, 178
PROTECT.DOCUMENT function 140, 178
PV function 37, 65-66

Question mark (?)
alert box icon 146
interactive command macros 96, 97, 138
wildcard character 59, 69
Quotation marks *See* Double quotation marks; Single quotation marks

RAND function 33, 34, 67
Random number function *See* RAND function
RATE function 37, 65-66, 67
Recorder range 115-116
Recording
absolute references 117
command macro 86-87
function macro 103
procedure 115-117
relative references 117
ref
See also specific function
ARGUMENT function 133-134
cell value 143, 156
description 144
GOTO function 98, 125, 142, 165
macro calling 99, 126-127, 142, 144
offset reference 143, 172
relationships 143, 144, 179
#REF!
CALLER function error 150
HLOOKUP function error 50
INDEX function error 52, 53
ISERROR function value 54
macro sheet closed 123
OFFSET function error 172

Reference
A1 *See* A1.R1C1 function
absolute *See* Absolute reference recording
active cell 143, 145
argument *See* Reference argument
calling function 143, 150
external *See* External reference
offset 143, 172
passing 136
R1C1 *See* A1.R1C1 function
ref *See* ref
relative *See* Relative reference
result 135-136
selection 143, 183
Reference argument
See also Argument; ref
restrictions 133-134, 147
type 29, 30, 134-135, 136, 147-148
Relative Record command 117
Relative reference
array formula 12
column width 151
recording 117
RELREF function 143, 179
REMOVE.PAGE.BREAK function 140, 179
REPT function 37, 67
Result
action-taking functions 82, 127-128
arrays *See* Array result
conversion 111, 179
external reference 136
function macros 143, 179-180
multiple-value *See* Array result
single-value 8
type
default 135
determination 73
specification 111, 135, 143, 179-180
RESULT function
description 143, 179-180
form 135
location 132, 135, 179
omission 112, 135
result type specification 111, 135, 143, 179-180
RETURN function
description 180
macro stopping 89, 98, 116, 124, 127, 143, 180
overwrite 116
RETURN function *(continued)*
value array return 114
ROUND function 34, 67
ROW function 33, 38, 68
ROWS function
arguments 33
array size determination 134
description 68
smart macros 94-95
special-purpose function 38
Run command
command macro calling 121
command macro running 85, 89, 92
RUN function 141, 180-181

SAVE function 139, 181
SAVE.AS function 139, 181
Scatter chart 141, 164
Screen update 142, 157
Scrolling 142, 166-167
SEARCH function 37, 68-69
Searching
cells 142, 161
database 153
matching records 141, 153
text 37, 68-69
SECOND function 38, 50-51
SELECT function 142, 182-183
SELECT.CHART function 141, 183
SELECTION function
description 183
smart macro 94-95
special-purpose function 143
Selection reference 143, 183
SELECT.LAST.CELL function 139, 183
Semicolon (;), array row separation 11
Serial number 43-44, 71-72
Series 154
Series formula 18-19
SERIES function 18
Set Recorder command
availability 83
recorder range setting 87, 90, 115
SET.CRITERIA function 140, 183
SET.DATABASE function 140, 184
SET.NAME function
description 143, 184
function macro use 110
loop counter setup 129

SET.PAGE.BREAK function 140, 179
SET.PRINT.AREA function 140, 185
SET.PRINT.TITLES function 140, 185
SET.VALUE function
  argument value change 112
  array argument value change 133
  description 143, 185
  loop counter setup 129
SHOW.ACTIVE.CELL function 139, 186
SHOW.CLIPBOARD function 140, 186
SIGN function 34, 69
SIN function 27, 36, 69
Single quotation marks (' ') 123
Single-step calculation *See* Calculation
SIZE function 142, 186
Smart macro 94-95
SORT function 140, 187-188
Special-purpose functions 38, 143
SQRT function 34, 70
Square root 34, 70
Start Recorder command 87, 116
Statistical functions 34
STDEV function 33, 34, 70
STEP function 143, 188-189
Stop Recorder command 87, 116
Stopping macros *See* Macro
Storing macros 83, 118
STYLE function 140, 189
SUM function
  arguments 8, 31, 33, 70
  array summation 8
  description 27, 70-71
  statistical function 34
Supporting worksheet 169-170

TABLE function 140, 189-190
TAN function 36, 71
Text
  argument type 11, 29, 30, 31
  array constant 11
  conversion to number 37, 74
  number of characters 27, 36, 55
  repetition 37, 67
  searching *See* Searching
  type number 73
TEXT function 37, 71
Text functions 36-37
TIME function 38, 71-72
Title printing 185
Tone *See* Bell
TRANSPOSE function
  arguments 33
  description 72-73
  result 16
  special-purpose function 38
TREND function
  arguments 33
  description 72
  result 16, 31
  statistical function 34
Trigonometric functions 35-36
TRUE function 33, 36, 73
TRUE value *See* Logical value
TYPE function 38, 73, 134

UNDO function 139, 190
UNLOCKED.NEXT 142, 191
UNLOCKED.PREV 142, 191

Value
  argument *See* Value argument
  consolidation 18
  constant *See* Constant value
  error *See* Error value
  function *See* SET.VALUE function
  logical *See* Logical value
  storing 18
  type 73
#VALUE!
  argument type error 111, 134, 148
  function macro argument error 136
  HLOOKUP function error 50
  INDEX function error 52, 53
  ISERROR function value 54
  result type error 111, 135
  SEARCH function error 68
  VALUE function error 74
Value argument
  description 30
  entry 112
  type 29, 30
VALUE function 37, 74
VAR function
  arguments 33, 74
  description 74-75
  statistical function 34
Vector argument 29, 30

VLINE function 142, 166
VLOOKUP function 38, 50
VPAGE function 142, 166
VSCROLL function 142, 166-167

WEEKDAY function 38, 45
WHILE-DO loop 130
Wildcard character 59, 69
Window
  activating 141, 145
  closing 141, 151
  moving 142, 171
  name listing 143, 191
  new 172
  scrolling 142, 166-167
  sizing 142, 186
Window menu 140
WINDOWS function 143, 191
Worksheet
  active 117-118
  consolidation 18
  function macro calling 107-109
  macro sheet differences 83, 91, 98, 123
  master worksheet 18
  saving 181
  supporting worksheet 169-170
Worksheet function
  See also Function
  ABS 33, 39
  ACOS 35, 39
  AND 32, 36, 39
  AREAS 32, 38, 40
  arguments *See* Argument
  ASIN 35, 40
  ATAN 35, 40
  ATAN2 35, 41
  AVERAGE 27, 32, 34, 41
  categories 25, 33-38
  CHOOSE 36, 42
  COLUMN 32, 38, 42
  COLUMNS
    arguments 32
    array size determination 134
    description 42
    smart macros 94-95
    special-purpose function 38
  COS 35, 43
  COUNT 32, 34, 43
  DATE 37, 43-44
Worksheet function *(continued)*
  DAVERAGE 35, 44-45
  DAY 37, 45
  DCOUNT 35, 46
  defined 25
  DMAX 35, 46
  DMIN 35, 46
  DOLLAR 36, 47
  DSTDEV 35, 47
  DSUM 35, 47
  DVAR 35, 48
  EXP 33, 48
  FALSE 32, 36, 48
  FIXED 36, 49
  FV
    arguments 31, 65-66, 106, 109
    description 49, 65-66, 106
    financial function 37, 106-110
  GROWTH 32, 34, 49
  HLOOKUP 38, 50
  HOUR 37, 50-51
  IF 36, 51, 94-95
  INDEX
    arguments 32
    array values 133
    description 51-53
    function macro use 114
    reference results 136
    special-purpose function 38
  INT 33, 53
  IRR 37, 53-54
  ISERROR 36, 54
  ISNA 36, 54
  ISREF 32, 36, 55
  LEN 27, 36, 55
  LINEST
    arguments 32
    description 55-56
    result size 16
    statistical function 34
  LN 34, 56
  LOG10 34, 56
  LOGEST 32, 34, 57
  LOOKUP 38, 57-58
  MATCH 38, 58-59
  MAX 32, 34, 60
  MID 36, 60
  MIN 32, 34, 61
  MINUTE 37, 50-51
  MIRR 31, 37, 61-62

Worksheet function *(continued)*
MOD 34, 62
MONTH 37, 45
NA 32, 38, 62
NOT 36, 63
NOW 32, 37, 63
NPER 37, 63, 65-66
NPV 37, 63-64
OR 33, 36, 64
PI 33, 34, 64
PMT 37, 65-66
PV 37, 65-66
RAND 33, 34, 67
RATE 37, 65-66, 67
REPT 37, 67
ROUND 34, 67
ROW 33, 38, 68
ROWS
  arguments 33
  array size determination 134
  description 68
  smart macros 94-95
  special-purpose function 38
SEARCH 37, 68-69
SECOND 38, 50-51
SIGN 34, 69
SIN 27, 36, 69
smart macros 94-95

Worksheet function *(continued)*
SQRT 34, 70
STDEV 33, 34, 70
SUM
  arguments 8, 31, 33, 70
  array summation 8
  description 27, 70-71
  statistical function 34
TAN 36, 71
TEXT 37, 71
TIME 38, 71-72
TRANSPOSE 16, 33, 38, 72
TREND
  arguments 33
  description 72
  result 16, 31
  statistical function 34
TRUE 33, 36, 73
TYPE 38, 73, 134
VALUE 37, 74
VAR 33, 34, 74-75
VLOOKUP 38, 50
WEEKDAY 38, 45
YEAR 38, 45

YEAR function 38, 45

MICROSOFT®
10700 Northup Way, Bellevue, WA 98004

Software Problem Report

Name _____

Street _____

City _____ State _____ Zip _____

Phone _____ Date _____

Instructions

Use this form to report software bugs, documentation errors, or suggested enhancements. Mail the form to Microsoft.

Category

_____ Software Problem

_____ Software Enhancement

_____ Documentation Problem (Document #_____)

_____ Other

Software Description

Microsoft Product _____

Rev. _____ Registration # _____

Operating System _____

Rev. _____ Supplier _____

Other Software Used _____

Rev. _____ Supplier _____

Hardware Description

Manufacturer _____ CPU _____ Memory _____ KB

Disk Size _____" Density: Sides:

Single _____ Single _____

Double _____ Double _____

Peripherals _____

Problem Description

Describe the problem. (Also describe how to reproduce it, and your diagnosis and suggested correction.) Attach a listing if available.

Microsoft Use Only

Tech Support _____ Date Received _____

Routing Code _____ Date Resolved _____

Report Number _____

Action Taken:

MICROSOFT®

Microsoft Corporation
10700 Northup Way
Box 97200
Bellevue, WA 98009

0186 Part No. 065-096-066